MASTER EN SCIENCES INFORMATIQUES

Re-ingénierie d'un système d'aide à la décision du raisonnement hybride au raisonnement par règle

Dallapé, Raja

Award date:
2006

Re-ingénierie d'un système d'aide à la décision : Du raisonnement hybride au raisonnement par règle

Raja Dallapé

Mémoire présenté pour l'obtention du diplôme de
Maître en informatique

Année académique 2005 – 2006

Résumé

La ré-ingénierie d'un système d'aide à la décision doit permettre d'améliorer celui-ci d'un point de vue général. Dans ce mémoire, nous présentons comment à partir d'une analyse de faiblesses d'un logiciel, par l'intermédiaire de ses interfaces, un choix de ré-ingénierie large a été opté pour améliorer globalement un logiciel d'aide à la décision. Ce processus de ré-ingénierie a été appliqué au système d'aide à la décision SMXpert provenant d'un processus de restructuration d'un autre système d'aide à la décision : COSMICXpert. Ces deux systèmes sont le fruit des laboratoires de recherche en génie logiciel de l'ÉTS (École de Technologie Supérieure).

Abstract

The software reengineering of a decision support software DSS would generally improve this one. In this thesis, we present how, starting from an analysis of the software's weaknesses, through its interfaces, the choice of a large reengineering has been made to improve a decision support software. This reengineering process has been applied to *SMX*^{pert} issued from a restructuration process of another decision support software : COSMICXpert. Both systems had been developed in the software engineering research laboratory of ÉTS (École de Technologie Supérieure).

Avant-propos

Ce document est l'aboutissement d'un stage de fin d'études à l'École de Technologie Supérieure (ÉTS) de Montréal, Canada. La réflexion et le travail fournis sont le fruit d'un étudiant en dernière année de maîtrise en informatique aux Facultés Notre-Dame de la Paix (FUNDP) de Namur, Belgique. Les résultats obtenus n'auraient pas été possibles sans le transfert de connaissances et la collaboration entre ces deux universités.

La rédaction de ce document n'est pas seulement l'œuvre d'un seul étudiant, c'est pourquoi nous voudrions remercier toutes les personnes qui, de près ou de loin, ont permis la réalisation de ce travail.

Nous remercions tout d'abord le Professeur Naji Habra, promoteur de ce mémoire, pour ses encouragements tout au long du stage et son aide à la rédaction de ce mémoire.

Nous remercions également les Professeurs Alain April et Jean-Marc Desharnais, tous deux maîtres de stage, pour leur aide précieuse et leur perpétuel suivi, autant sur place qu'outre-atlantique.

Nous remercions, bien entendu, tous nos proches, pour leur soutien et leur patience. Et un remerciement particulier est adressé à Monsieur Denis Urbain qui a consacré du temps et de l'énergie au projet SMX^{pert}.

Table des matières

Table des illustrations

Introduction

Ce mémoire traite de l'amélioration d'un logiciel d'aide à la décision dans le domaine de la maintenance. Il explique comment à partir d'une analyse d'interface, des problèmes de diverses natures ont été mis à jour et la décision qui a été prise pour améliorer le logiciel.

Contexte

La maintenance de logiciel est un domaine qui devient de plus en plus vaste et de plus en plus présent en industrie. Elle n'en est plus à son balbutiement, et possède son propre processus et sa propre gestion stratégique afin de corriger des erreurs ou apporter des améliorations pour que les logiciels continuent à répondre de la manière la plus efficiente possible aux besoins de ses utilisateurs.

SM^{Xpert} est un logiciel ayant pour but d'aider les ingénieurs en maintenance dans leurs activités. Il se fonde sur les travaux du professeur Alain April de l'École de Technologie Supérieure (ÉTS) de Montréal. Dans le cadre d'une recherche doctorale, le professeur a proposé, sur base d'une liste de problèmes liés à la maintenance logicielle, un modèle de la maturité des processus nommé « Software Maintenance Maturity Model » (S^{3m}). Il permet d'évaluer et d'améliorer le processus de maintenance.

Le logiciel SM^{Xpert} est un logiciel créé à partir d'une restructuration d'un autre logiciel (le logiciel COSMICXpert), ayant aussi pour but l'aide à la décision. Le problème est que ce dernier est très complexe à utiliser pour l'expert désirant entrer ou modifier des informations dans la base de connaissance. Certains utilisateurs, dont des experts en maintenance, révèlent qu'il est impossible d'entrer des informations, ou en tout cas qu'ils y sont parvenus en contournant les interfaces, c'est-à-dire en accédant directement aux fichiers XML dans lesquels la connaissance est mémorisée. Il est donc nécessaire d'améliorer ce logiciel afin de rendre possibles ces fonctionnalités d'évolution de la base des connaissances.

Ce projet a pour objectif principal de proposer des solutions d'amélioration du logiciel SM^{Xpert}.

Problématique

Le logiciel SM^{Xpert} se révèle très difficile à l'usage. Les utilisateurs ne parviennent pas à dominer correctement les fonctionnalités d'évolution ou de modification de la base des connaissances. Le symptôme premier est la difficulté d'utilisation des interfaces ; nous savons que les interfaces sont une partie cruciale pour la facilité d'utilisation d'un logiciel.

La problématique étudiée dans ce document est donc l'amélioration d'un système expert, à partir d'une analyse d'interfaces, en y amenant une solution de ré-ingénierie.

Objectifs

La motivation initiale constituait à améliorer l'interface des utilisateurs de type « expert » du logiciel SM^{Xpert}. Mais l'objectif s'est élargi, après une familiarisation avec le logiciel, devenant une amélioration générale de ce dernier, avec par exemple une amélioration du modèle conceptuel du logiciel ou encore le remplacement de fichier XML par un système de gestion de base de données.

Historique

La recherche en génie logiciel a permis de mettre à jour la méthode fonctionnelle COSMIC-FFP (COSMIC étant l'acronyme de Common Software Measurement International

Consortium, FFP celui de Full Functional Point). COSMIC-FFP vise à améliorer la mesure des logiciels et est devenue en décembre 2002 une norme ISO/IEC avec pour numéro : 19761.

Ensuite, un système à base de connaissances pour l'apprentissage de la méthode est né. Le projet de création du logiciel COSMICXpert venait de voir le jour. La première version de COSMICXpert a été développée par Tim Küssing, un étudiant allemand, lors de son stage à Montréal avec le professeur Desharnais. Mais cette première version présentait deux limites :

- le logiciel était en mode local, ce qui veut dire qu'il était difficilement accessible aux mesureurs ;
- les données ne pouvaient pas être validées facilement car un grand nombre de fichiers au format texte devaient être créés pour ne mémoriser que très peu d'information de la base de connaissances.

Une seconde version a été créée pour permettre l'accessibilité du logiciel à plus grande échelle, ainsi qu'un système de mémorisation de données plus efficace. C'est le fruit du travail de François Gruselin et Julien Vilz, en automne 2002. Messieurs F. Gruselin et J.Vilz qui ont proposé une solution WEB, pour rendre le logiciel accessible à un plus grand nombre d'utilisateurs. Ensuite le logiciel utilisait des fichiers XML, pour réduire le nombre de fichiers servant à la mémorisation.

La seconde version de messieurs F. Gruselin et J. Vilz assurait déjà tous les besoins des mesureurs, il restait encore à rendre possible l'évolution de la base des connaissances.

Ce sont les stagiaires Christophe Duterme et Nicolas Fabry qui ont terminé la deuxième version du logiciel COSMICXpert en automne 2003. Ils ont également ajouté un mécanisme permettant de gérer la concurrence lors de l'utilisation des fonctionnalités d'évolutions, ou modification de la base des connaissances. L'intégrité de la base des connaissances était ainsi mieux préservée.

COSMICXpert était enfin complet et accessible depuis l'Internet. Mais le projet s'étant étalé sur plusieurs années, la documentation et le code source étaient trop hétérogènes, il n'y avait plus vraiment d'architecture calquée sur un modèle précis. Enfin, beaucoup d'erreurs subsistaient.

En 2004, les stagiaires Stéphane Sandron et Benoît Vanderose ont tout d'abord refait la documentation de COSMICXpert pour la rendre plus homogène, ensuite ils ont donné une architecture de type « Modèle - Vue - Contrôleur » et finalement ils ont corrigé la majeure partie des erreurs du logiciel COSMICXpert.

Il reste néanmoins encore beaucoup à faire. Toutes les erreurs n'ont pas été corrigées et les interfaces sont peu intuitives ; elles ont été modifiées lorsque des ajouts ont été faits, mais n'ont jamais réellement fait l'objet d'une réflexion.

Enfin, les stagiaires Sandron et Vanderose ont, à partir du logiciel COSMICXpert, appliqué un processus de restructuration pour donner naissance à SM^{Xpert}. SM étant l'acronyme de Software Maintenance, le nom du logiciel indique presque déjà son utilité. En effet, son domaine d'application est la maintenance de logiciel, et il permettra d'aider un utilisateur à évaluer la maturité d'un processus de maintenance et de résoudre les problèmes de celui-ci.

Contenu et organisation

Cet ouvrage se divise en deux grandes parties, la première partie est une partie théorique avec premièrement une revue de littérature couvrant les références nécessaires à la compréhension des notions utilisées tout au long de ce document. Ainsi sont présentées dans une première section les exigences ergonomiques pour le travail de bureau avec terminaux à écran

de visualisation, avec les principes d'ergonomie, des règles et des recommandations relatives à la présentation de l'information. Ensuite vient, dans une seconde section, une présentation des systèmes d'aide à la décision en général avec une distinction faite entre le raisonnement par cas et le raisonnement par règle. Et enfin, dans une dernière section, une présentation de la ré-ingénierie et de la restructuration. Après cette revue de littérature, la première partie de l'ouvrage se termine par une présentation générale de la maintenance de logiciel.

La seconde partie est plus pratique. Elle commence par un premier chapitre décrivant le logiciel à améliorer ; le chapitre suivant offre une analyse des faiblesses de ce logiciel ; les deux derniers chapitres proposent respectivement une ré-ingénierie du logiciel et les premiers pas vers une nouvelle version.

Partie I

Chapitre 1 Revue de littérature

Cette revue de littérature contient les références nécessaires dans le cadre de ce projet visant à améliorer le logiciel SM^{Xpert}.

Nous allons premièrement voir les meilleures pratiques d'interface. Ensuite, comme le projet vise à améliorer un système expert en général, les références en matière de systèmes experts, système d'aide à la décision, ré-ingénierie et restructuration sont également présentées.

1.2 Les exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation

1.2.1 Introduction

Les exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation sont une source d'information importante dans le cadre de ce projet car elles permettront dans un premier temps d'évaluer une interface et par la suite de servir d'outil pour améliorer cette même interface. Avant de voir les exigences proprement dites, il est tout d'abord intéressant d'expliquer les principes généraux.

Les principes généraux d'ergonomie seront donc tout d'abord présentés, suivis des règles d'ergonomie découlant de ces principes et permettant à un utilisateur de se familiariser rapidement au dialogue avec une interface.

Ensuite, nous verrons les recommandations relatives à la présentation de l'information et comment concevoir des objets graphiques suivant les attentes des utilisateurs.

Tous ces principes, recommandations et règles peuvent également être appliqués à l'occasion de la spécification, du développement ou de l'évaluation d'un logiciel.

1.2.2 Principes d'ergonomie

Ces principes ont pour but de rendre une interface plus adaptée à l'utilisateur, de réduire la charge cognitive et minimiser l'effort de mémorisation. Les principes ont deux autres buts, plus vastes encore. Ils veulent minimiser le multitâches et permettre d'avoir un modèle mental clair d'une application.

Ces principes doivent prendre en compte les caractéristiques de l'utilisateur tels que : la capacité de concentration, les limites de la mémoire à court terme, les comportements

d'apprentissage, le degré d'expérience dans l'activité du logiciel, la compréhension par l'utilisateur de la structure sous-jacente et du but du système avec lequel il va dialoguer.

Mais ces principes ne sont pas indépendants il est même probable de devoir faire un compromis entre avantages et inconvénients.

Enfin, il faut prendre en compte le champ d'application spécifique : les buts de l'organisation, des besoins du groupe (final) d'utilisateurs concernés, les tâches que l'application doit permettre de réaliser et les technologies et ressources disponibles.

Voici la liste des principes [8] et [9] :

Adaptation de la tâche : « Un dialogue est considéré comme adéquat, pour une tâche, lorsqu'il permet à l'utilisateur de réaliser cette tâche de façon efficace et efficiente. » [9]

Caractère auto descriptif : « Un dialogue est auto descriptif lorsque chaque étape du dialogue est immédiatement compréhensible grâce au retour d'information du système, ou est expliquée à l'utilisateur à sa demande. » [9] Il faut aussi informer l'utilisateur sur l'état d'un système, c'est-à-dire lui indiquer qu'une action risque de prendre du temps, lui indiquer si des données entrées ont été bien insérées, etc.

Contrôle explicite : « Ce principe signifie que même si c'est le logiciel qui a le contrôle, l'interface doit apparaître comme étant sous le contrôle de l'utilisateur et surtout exécuter des opérations uniquement à la suite d'actions explicites de l'utilisateur. » [8]

Conformité aux attentes de l'utilisateur : « Un dialogue est conforme aux attentes de l'utilisateur lorsqu'il est cohérent et correspond aux caractéristiques de l'utilisateur, telles que la connaissance de la tâche, la formation, l'expérience et les conventions communément admises. » [9]

Facilité à l'apprentissage : « Un dialogue permet l'apprentissage lorsqu'il soutient et guide l'utilisateur dans l'apprentissage de l'utilisation du système. » [9]

Gestion des erreurs : « Un dialogue est tolérant aux erreurs si, malgré des erreurs d'entrée évidentes, le résultat prévu peut être obtenu soit sans action corrective, soit avec une action corrective minimale de la part de l'utilisateur. » [9]

Flexibilité : « Un dialogue permet l'individualisation lorsque l'interface logicielle peut être modifiée pour s'adapter aux besoins de la tâche, aux préférences individuelles et aux compétences de l'utilisateur. » [9] Une application doit pouvoir s'adapter à l'évolution cognitive d'un utilisateur et également à divers utilisateurs.

Cohérence : Ce qui s'affiche sur un écran doit être cohérent avec le monde réel, et le vocabulaire doit être le vocabulaire usuel d'un utilisateur. Exemple : si une application consiste à introduire les données d'une personne, il est préférable que ce qui s'affiche à l'écran soit directement perçu comme un formulaire. Il doit aussi y avoir une cohérence entre les données affichées à l'écran. Exemple : une uniformité dans les boutons de commande.

Concision : Eviter les manipulations trop longues afin de ne pas fatiguer un utilisateur. Exemple : trop d'étapes dans une procédure. Dans ce cas, l'utilisation de valeur par défaut ou de macros est intéressante.

A. Règles d'ergonomie

Voici à présent les règles proprement dites, chacune d'entre elles étant présentée en rapport avec le principe qu'elle tend à faire respecter. Il faut néanmoins préciser que certains principes sont suffisamment explicites pour ne pas nécessiter de règle.

Il existe de nombreuses règles d'ergonomie, plus de 3000 [8]. Elles ne sont évidemment pas toutes applicables à chaque application ou utilisateur. Nous allons énoncer les règles permettant de souligner les points faibles, en matière de simplicité et d'intuitivité, d'une

interface.

A.1 Adaptation de la tâche

Une interface permettant d'accomplir une tâche ne doit présenter que l'information permettant de l'exécuter.

Il faut que l'utilisateur puisse toujours avoir accès à un menu d'aide expliquant la tâche qu'il désire exécuter.

Toute action pouvant être automatisée doit être faite par le logiciel ; il convient aussi de prendre en compte la complexité de la tâche (ex : utiliser un menu offrant les choix possibles lorsqu'il y a un ensemble d'entrées d'alternatives). S'il existe des possibilités de données d'entrée par défaut, l'utilisateur ne doit pas être obligé d'entrer ces valeurs.

A.2 Caractère auto descriptif

S'il y a lieu, le logiciel se doit de fournir un compte rendu des actions de l'utilisateur, qui se réfèrent strictement à la situation pour laquelle il est nécessaire. Et « si des conséquences graves peuvent résulter d'une de ses actions, une explication doit lui être fournie ainsi qu'une demande de confirmation. » [9]

Ce compte rendu et ces explications doivent être adaptés au niveau de connaissance de l'utilisateur et lui permettre d'acquérir une compréhension globale du logiciel.

Le compte rendu et les explications se doivent d'être plus ou moins détaillées selon les besoins de l'utilisateur (exemple : en appuyant une fois sur la touche « Aide », l'utilisateur obtient une brève explication ; en appuyant deux fois, il obtient une explication détaillée).

Lorsqu'il faut entrer des données, il convient que le logiciel indique à l'utilisateur la nature des données à introduire.

Il faut aussi informer l'utilisateur sur l'état d'un système, c'est-à-dire lui indiquer qu'une action risque de prendre du temps.

A.3 Contrôle explicite

L'utilisateur doit donc contrôler le déroulement du dialogue, « si le dialogue a été interrompu il convient que l'utilisateur ait la possibilité de déterminer à quel endroit le reprendre chaque fois que la tâche le permet. » [9]

L'utilisateur doit pouvoir annuler au moins la dernière étape d'un dialogue.

Enfin, les méthodes d'interaction doivent être adaptées aux besoins et aux caractéristiques de l'utilisateur.

A.4 Facilité d'apprentissage

« Il convient que l'utilisateur dispose des règles et concepts sous-jacents utiles à l'apprentissage, afin de pouvoir constituer ses propres stratégies de regroupement et ses propres règles de mémorisation. » [9]

A.5 Tolérance aux erreurs

L'application doit assister l'utilisateur dans la prévention et la détection d'erreurs d'entrée. Le système doit aussi empêcher qu'une erreur d'entrée provoque des résultats non prédéfinis et des défaillances du système.

« Dans le cas où le système peut corriger des erreurs automatiquement, il convient que le système avertisse l'utilisateur de l'exécution des corrections et qu'il laisse la possibilité d'ignorer les corrections. » [9]

A.6 Flexibilité

Il convient que le logiciel permette à l'utilisateur de choisir entre différentes formes de représentation, selon ses préférences et la complexité de l'information à traiter.

1.2.3 Recommandations relatives à la présentation de l'information

La présentation de l'information est aussi importante que le dialogue car il le facilite. L'un ne va pas sans l'autre. L'information doit présenter certaines caractéristiques permettant un maximum de tâches de perception de manière efficace, effective et satisfaisante.

Voici la liste de ces caractéristiques [10] :

- Clarté : « le contenu de l'information est transmis rapidement et avec précision. »
- Discernabilité : « l'information affichée peut-être distinguée avec précision. »
- Concision : « les utilisateurs reçoivent uniquement les informations nécessaires à l'exécution de la tâche. »
- Cohérence : « la même information est présentée de manière identique sur toute l'application, en toute conformité avec les attentes de l'utilisateur. »
- Détectabilité : « l'attention de l'utilisateur est dirigée vers l'information demandée. »
- Lisibilité : « information facile à lire. »
- Compréhensibilité : « le sens est clairement compréhensible, sans ambiguïté, interprétable et reconnaissable. »

A. Organisation de l'information.

C'est en organisant l'information qu'il est possible de respecter les caractéristiques présentées ci-dessus. L'organisation de l'information est facilitée par l'utilisation d'objets graphiques. Voici des recommandations relatives aux objets graphiques afin de respecter les caractéristiques nécessaires de l'information.

A.1 Les fenêtres

L'utilisation de fenêtres est plus appropriée lorsque :

- l'utilisateur surveille ou accède à plusieurs systèmes, applications ou processus simultanément ;
- l'utilisateur évalue, compare ou manipule plusieurs sources d'information ou plusieurs vues d'une source unique d'information ;
- l'utilisateur travaille souvent alternativement entre des tâches, des systèmes, des applications, des dossiers, des sections ou des vues ;
- l'utilisateur doit préserver le contexte d'une tâche plus vaste tout en effectuant des sous-tâches individuelles ;
- l'utilisateur doit prendre en charge des événements d'un système ou d'une application avant que les opérations de tâche primaire ne puisse continuer ;
- l'utilisateur doit avoir accès de façon occasionnelle à des composants de dialogue supplémentaires (ex : menu), situés à proximité du point de l'écran où se concentre l'activité en cours de l'utilisateur.

Lorsque plusieurs fenêtres sont utilisées il faut fournir une identification unique (ex : un nom de fenêtre), concevoir les positions et les dimensions des fenêtres par défaut de manière à

minimiser le nombre d'opérations que les utilisateurs doivent exécuter pour accomplir une tâche.

Lorsqu'une relation de subordination intervient entre fenêtres, les relations entre fenêtre principale et ses secondaires doivent toujours être visuellement apparentes.

« Il convient de pouvoir discerner visuellement les éléments de commandes de fenêtres exécutant différentes fonctions, et de les placer de manière cohérente et au même endroit dans chaque fenêtre. » [10]

« Si cela est approprié à la tâche, il convient d'autoriser les utilisateurs à choisir leur format de fenêtrage préféré et à le sauvegarder comme format par défaut. » [10]

A.2 Zones

« Les zones sont à placer de manière homogène, tout au cours du dialogue dans une application. » [10] « Le densité de l'information affichée doit être telle qu'elle ne puisse pas être perçue par l'utilisateur comme trop encombrée. » [10] Et si l'utilisateur doit discerner des relations entre des ensembles d'information affichés séparément, il est souhaitable d'afficher les deux ensembles d'information sur un seul écran.

Quant aux zones d'entrée/sortie, il convient, dans la mesure du possible, d'afficher dans la zone d'entrée/sortie toutes les informations exigées pour exécuter une tâche donnée. Si ce n'est pas possible, il convient :

1. de structurer les informations requises en sous-ensemble correspondant aux étapes des tâches ;
2. que ces sous-ensembles prennent en charge les sous-tâches appropriées et soient significatifs pour les utilisateurs prévus, et
3. que le partage de l'information n'entraîne aucune réduction des performances de la tâche.

A.3 Groupes

Le regroupement des informations sur l'écran aide l'utilisateur à percevoir, trouver et interpréter/comprendre plus facilement l'information.

« Il convient que les groupes soient perçus comme distincts de par l'espacement et l'emplacement. » [10]

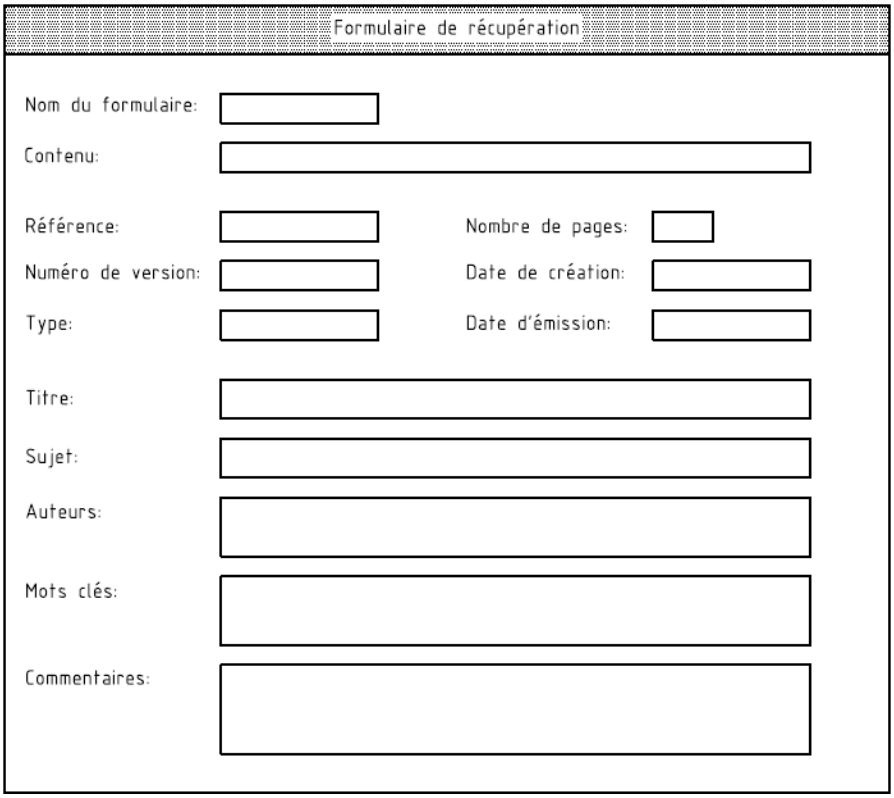


Figure 1 : exemple de regroupement [10]

Les lois suivantes facilitant l'utilisation de groupe.

Loi de rapprochement : les éléments très rapprochés dans l'espace sont perçus comme appartenant les uns aux autres.

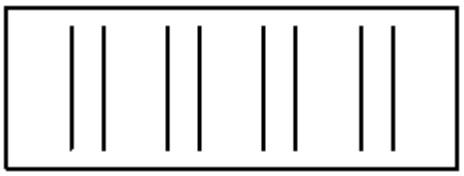


Figure 2 : illustration de la loi de rapprochement [8] et [10]

Loi de similarité : les éléments sont perçus comme appartenant les uns aux autres s'ils sont similaires. Dans l'exemple illustré l'observateur perçoit des colonnes et non des lignes.

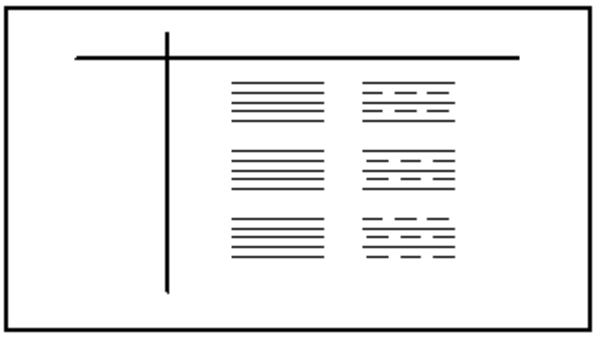


Figure 3 : illustration de la loi de similarité [8] et [10]

Loi de fermeture : les parties non existantes d'une figure sont ajoutées ou les figures incomplètes sont automatiquement complétées. C'est le cas lorsque des groupes d'informations sont séparés dans l'espace, l'observateur tente alors de créer une figure cohérente.

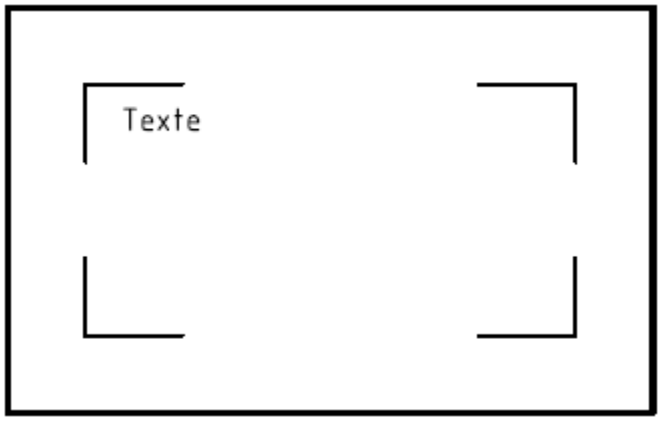


Figure 4 : illustration de la loi de fermeture [8] et [10]

A.4 Les listes

Utilisée aussi pour l'organisation de l'information, les listes doivent être organisées selon un ordre logique ou naturel adapté à la tâche (si pas possible, l'ordre alphabétique peut être envisagé).

« Les éléments et les groupes d'éléments d'une liste doivent être visuellement distincts afin de faciliter le balayage visuel. » [10]

Le format des listes d'informations alphabétiques doit dépendre des conventions linguistiques (ex : justifier à gauche les listes verticales d'informations alphabétiques pour les langues se lisant de gauche à droite). Pour les informations numériques, il est préférable de justifier à droite celles sans signe décimal (virgule ou point). Pour celles contenant des signes décimaux, il est préférable de les aligner par rapport au signe décimal.

Dans les listes numériques, il convient d'utiliser une taille de police fixe avec un espacement constant.

« Si une liste s'étend au-delà de la zone d'affichage disponible, il convient de fournir une indication de suite de liste. » [10]

A.5 Les tableaux

Utilisés pour l'organisation d'informations dans des sous-ensembles visuels ayant un sens.

« La disposition des informations dans les tableaux doit être telle que les données les plus pertinentes pour les utilisateurs ou revêtant la plus grande priorité soient affichées dans le colonne la plus à gauche et que les données associées, moins significatives, figurent dans des colonnes situées plus à droite. » [10]

« Si un tableau utilise des titres de colonnes et de rangées et qu'il s'étend au-delà de l'affichage disponible, alors les titres associés aux colonnes et/ou aux lignes visibles restent également visibles. » [10]

A.6 Les labels

Utilisés pour indiqués la signification des éléments d'information, lorsque le sens d'un élément d'écran n'est pas évident et clairement compréhensible pour les utilisateurs prévus.

Les labels doivent suivre une cohérence grammaticale (exemple : utilisation cohérente de combinaisons nom-verbe).

L'emplacement du label se doit aussi d'être homogène, à proximité de l'élément d'information désigné. Leur format et leur alignement doivent aussi être homogène.

A.7 Les champs

« Il convient tout d'abord de bien faire la distinction visuelle entre les champs de saisie et les champs en lecture seule (ex : couleur, format, …). Si la tâche l'exige il convient de pouvoir distinguer l'information saisie par l'utilisateur des informations générées par le système dans les champs de saisie. » [10]

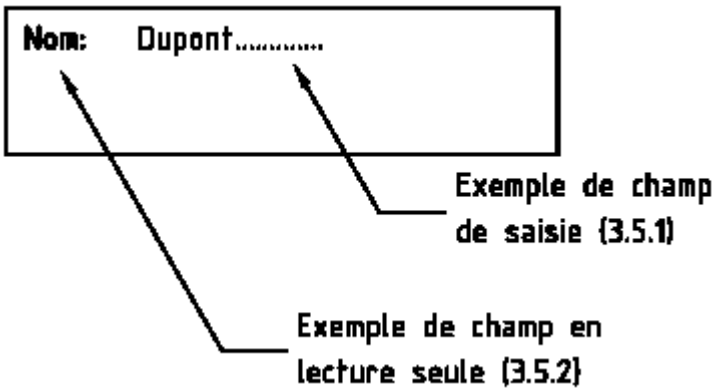


Figure 5 : exemple de champs [10]

Si un champ de saisie nécessite un format spécifique, il convient d'indiquer clairement les formats du champ de saisie, tandis que les champs de saisie fixes doivent être clairement indiqués.

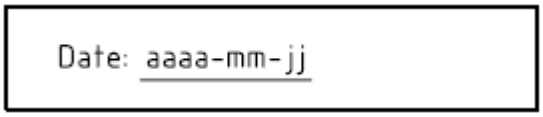


Figure 6 : exemple d'indication de format [10]

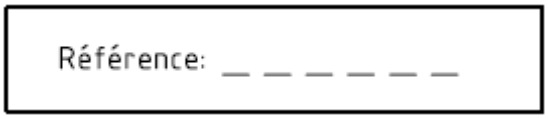


Figure 7 : exemple de longueur fixe indiquée [10]

A.8 Les techniques de codage

« Il faut concevoir les créations ou règles de code en impliquant les utilisateur prévus, et en prenant en compte leurs attentes et leurs tâches. » [10] Ceux-ci doivent avoir un sens, le niveau de signification augmente lorsqu'il existe des associations claires entre l'information codée et le sens prévu. Ainsi les codes mnémoniques sont conseillés plutôt que des codes arbitraires. Si ce n'est pas possible, alors le sens doit être facilement accessible.

Ces codes doivent être visuellement distincts les uns des autres et utilisés de manière constante avec le même sens ou la même fonction.

A.9 Marqueurs

« Les symboles particuliers également connus sous le nom de « marqueurs », sont utilisés pour attirer l'attention sur des éléments alphanumériques choisis » [10]. « Il est préférable d'utiliser des marqueurs différents pour indiquer la sélection unique et la sélection multiple. » [10]

Les marqueurs doivent être près de l'élément marqué, mais de manière à ce qu'ils n'apparaissent pas comme faisant partie des éléments affichés.

Figure 8 : exemple de marqueur [10]

1.2.4 Sommaire

Ainsi, si une interface s'adapte à la tâche d'un utilisateur et qu'elle répond à ses attentes en lui fournissant l'information nécessaire à l'exécution de sa tâche, en utilisant son vocabulaire et en restant cohérente tout au long d'un dialogue ; si l'utilisateur peut contrôler son dialogue et recevoir des comptes rendus de ses actions ainsi qu'un soutient et une guidance, il peut mieux comprendre le dialogue avec le terminal qu'il utilise et arrive plus facilement à son but.

Ces principes rendent donc une interface plus intuitive et plus facile d'utilisation et permettent à l'utilisateur d'accomplir une tache plus rapidement et correctement. De plus, si un système est renforcé en le rendant capable de supporter et de prévenir les erreurs de l'utilisateur, ce dernier permettra d'accélérer encore davantage la réalisation d'une tâche. Et finalement, si le système peut être individualisé, il sera encore plus simple pour l'utilisateur de le comprendre et de l'utiliser.

En addition avec ces principes, des objets graphiques conçus de manière cohérente et de façon à rendre leur utilisation plus naturelle ou à faciliter la lecture de l'information qu'ils contiennent rendent également le dialogue plus intuitif et l'exécution d'une tâche plus facile. Si l'information présentée est plus claire, plus concise et compréhensible, cela permet déjà à l'utilisateur de réaliser plus aisément des tâches de perception. De plus, une bonne organisation de l'information, tant dans le fenêtrage que pour la résolution de l'écran, permet d'éviter à l'utilisateur bien des manœuvres inutiles qui favorisent la déconcentration.

Ces principes et recommandations peuvent donc déjà permettre une critique du logiciel existant.

1.3 Les systèmes experts et systèmes informationnels d'aide à la décision

1.3.1 Introduction

Depuis l'avènement de l'informatique, l'homme a cherché à créer des système pouvant supporter tant ses prises de décision que la gestion de ses activités. Au fil du temps, un continuum d'expertise s'est installé, allant des systèmes classiques de gestion aux systèmes experts.

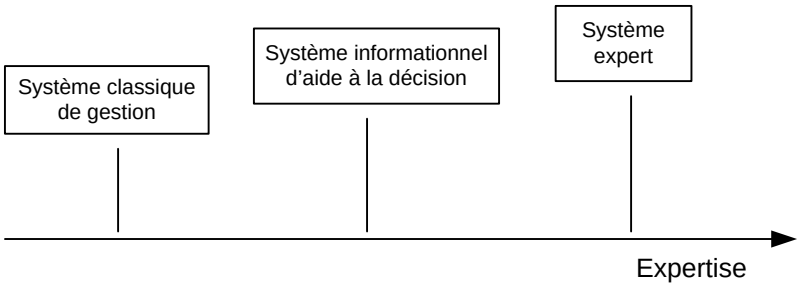


Figure 9 : continuum d'expertise [1]

Nous allons tour à tour présenter les concepts reliés à un système informationnel d'aide à la décision et les systèmes experts. Il existe plusieurs types de systèmes experts se différenciant par leur type de raisonnement. On distingue le raisonnement basé sur les règles et le raisonnement basé sur les cas.

1.3.2 Système informationnel d'aide à la décision

Un système d'aide à la décision est un système d'information, permettant de supporter des prises de décisions pour des situations dans lesquelles il n'est pas possible ou souhaitable d'automatiser entièrement le processus de décision [1].

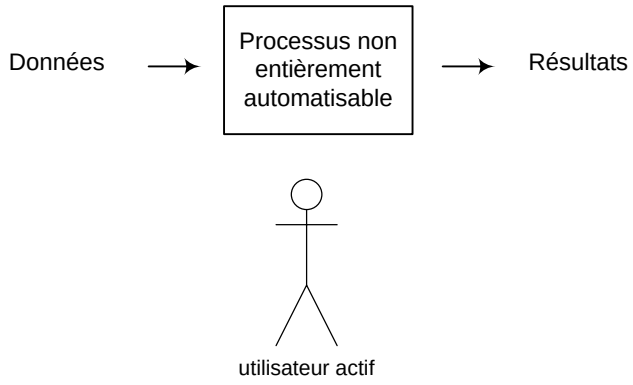


Figure 10 : système informationnel d'aide à la décision [1]

Un processus non entièrement automatisable est un processus dans lequel il doit y avoir une interaction, en l'occurrence ici pour la résolution d'un problème, entre l'utilisateur et le système.

Un système informationnel d'aide à la décision devrait idéalement être vu comme une « boite noire » avec laquelle un utilisateur peut communiquer aisément par un langage proche du langage naturel.

1.3.3 Systèmes experts

Un système expert est un outil pouvant reproduire des mécanismes cognitifs, dans un domaine précis, à partir d'une base de connaissance et d'un moteur d'inférence. C'est un logiciel simulant le mode de raisonnement d'un expert, pour acquérir de nouvelles connaissances à partir de faits et règles connus. Les systèmes experts sont des outils d'aide à la décision.

Le but des systèmes experts est d'obtenir l'expérience et l'expertise dans un domaine particulier. Les systèmes experts se veulent aussi évolutifs grâce à un module d'acquisition des connaissances. L'acquisition de connaissances ne signifie pas uniquement ajouter de nouvelles connaissances, mais aussi mettre en relation des nouvelles connaissances entre elles et avec les anciennes connaissances. Cette démarche permet d'éviter d'écrire de nouveaux programmes. Une base de connaissance permet de mémoriser tout cela.

« La base de connaissance est un élément capital, c'est la représentation des connaissances de l'expert et la description d'heuristiques utiles. » [*Web02*]

« Les systèmes experts sont aussi capables de résoudre d'eux-mêmes les problèmes, de prendre des décisions et de raisonner sur des problèmes. » [1] La résolution se fait par manipulation de connaissances ou par déduction. Ainsi il existe plusieurs types de systèmes experts, chacun réglant des problèmes d'une manière qui lui est propre. Ce sont ces différentes façons de régler les problèmes qui classifient les systèmes experts. On distingue le raisonnement par cas et le raisonnement par règle.

A. Le raisonnement basé sur les cas

Le raisonnement basé sur les cas, le « case based reasonning » est une technique de modélisation du raisonnement qui se sert de l'importance du passé pour résoudre un problème. C'est une conséquence simple de la nature humaine. Il n'est pas rare d'entendre : « évitons un nouveau … » qui est ici une référence négative, mais il existe évidemment des références positives. Cette technique de modélisation du raisonnement à partir de précédent est fortement utilisée en psychologie cognitive, en enseignement et autres professions comme le droit, pour la résolution de cas juridique, grâce à la jurisprudence qui a un rôle important.

« Le précédent sert de cadre de réflexion » [*Web03*] et est très intéressant pour de nouveaux problèmes non encore résolus.

« Le "case based reasoning" est une des techniques de raisonnement analogique. Il signifie raisonner à partir de cas ou d'expériences anciennes pour résoudre un problème, critiquer des solutions, expliquer des situations anormales ou interpréter des situations » [*Web01*].

Une des pionnières de cette technique de modélisation est Janet Kolodner. Voici sa définition : « Cased-based reasoning can mean adapting old solutions to meet new demands, using old cases to explain new situations, using old cases to critique new solutions, or reasoning from precedents to interpret a new situation (much like lawyers do) or create an equitable solution to a new problem (much like labour mediators do) » [*Web01*].

Abran et Desharnais [2] présentent les suppositions de J. Kolodner quant à l'utilisation du raisonnement par cas :

- Régularité : le monde est essentiellement régulier et prévisible. Les mêmes actions faites dans les mêmes conditions donneront les mêmes résultats (ou très similaires).
- Répétition : les événements tendent à se répéter. Raisonner par cas est donc très susceptible d'être utile dans le futur.
- Consistance : des petits changements de l'environnement requièrent des petits changements de raisonnement et aussi dans la solution.

Il existe deux styles de raisonnement à base de cas :

1. le style résolution de problème, à partir d'anciennes solutions, utilisées comme guide. Ce raisonnement est utile dans la résolution de problèmes comme par exemple la planification de tâches ;
2. le style interprétatif qui interprète une nouvelle situation dans le contexte d'une ancienne situation. Ce type de raisonnement est intéressant pour l'argumentation, peser le pour et le contre.

Ces deux styles sont forts dépendants de la capacité humaine à la recherche de cas appropriés et aussi à l'interprétation humaine. Il est donc nécessaire de bien classer les cas, et ce n'est pas un exercice évident.

Par contre, l'être humain est plus à l'aise pour le raisonnement analogique. En effet, lorsqu'il y a une part d'expérience pour une situation nouvelle mais comportant des similarités, les cas du passé « transmettent » leur logique pour la nouvelle situation. Même un novice peut se servir des cas, la justification sera plus difficile. La résolution, grâce au raisonnement basé sur des cas, est performante lorsque l'utilisateur a une certaine expérience dans le domaine.

Un ordinateur résolvant des problèmes par analogie est appelé un « Case base reasonning system » (CBR Systems) ; il est doté d'une base de connaissance qui est constituée des cas. Les CBR Systems sont des systèmes particuliers de raisonnement et ont une diversité d'application dans beaucoup de domaines.

A.1 Qu'est-ce qu'un Case-Based Reasonning tool

« Sur le plan cognitive un Case-Based Reasonning tool (*CBR tool*) est un outil capable d'utiliser une connaissance spécifique venant de situations problématiques expérimentées auparavant.

[…] Un CBR tool est aussi une approche incrémentale d'apprentissage. Une nouvelle expérience est retenue à chaque fois qu'un problème est résolu, le rendant immédiatement disponible pour des problèmes futurs. » [2]

A.2 Le processus de raisonnement par cas

Le processus principal d'un raisonnement par cas se décompose en 4 étapes [3] :

Étape 1 : Les cas sont organisés dans la base des connaissances. Il faut, pour résoudre les problèmes, un nombre initial de cas problèmes dans la base des connaissances. Ces cas initiaux doivent être indexés et organisés en une structure utilisable. C'est par l'utilisation de cette structure que l'utilisateur va, dans la première étape chercher les cas les plus appropriés. Par exemple grâce à l'analyse multicritères.

Etape 2 : Choisir le meilleur cas et l'adapter au cas courant ou utiliser sa solution comme guide.
Dans le contexte d'une adaptation d'un cas au cas courant, il se peut qu'un cas corresponde exactement. Pour ce qui est de l'utilisation de la solution comme guide, il faut en réalité construire la solution pour le nouveau cas.

Etape 3 : Tester la nouvelle solution. Dans cette étape la solution du cas problème en cours est testée. C'est à l'utilisateur de décider si la solution est acceptable pour son cas problème.

Etape 4 : Ajouter une nouvelle connaissance à la base de connaissance. Dans cette étape, l'utilisateur décide si la solution trouvée à l'étape précédente devrait être ajoutée à la base de connaissance pour en faire une future référence.

A.3 Les avantages du raisonnement par cas

Voici la liste des avantages cités dans la littérature concernant le raisonnement par cas :

- Résolution de problèmes avec un domaine partiellement connu. En effet, lorsque le domaine est partiellement défini, le raisonnement par cas est plus approprié. En d'autres mots, les domaines traités par les CBR systems sont des domaines imparfaitement définis mais riches en expérience ;
- Plus approprié pour des problèmes nouveaux ;
- Proche du raisonnement humain. Il est souvent naturel de résoudre un problème en se servant de sa propre expérience ;
- Raisonnement efficace. L'utilisateur se focalise sur les aspects, de la résolution d'un problème, qui sont importants. De plus le CBR system aide à éviter de prendre une mauvaise direction qui pourrait engendre de nouveaux problèmes ;
- Acquiert rapidement des connaissances. La grande partie des connaissances se trouvant dans les cas, collecter des cas n'est pas une tâche difficile et beaucoup de domaines ont déjà des cas existants ;
- Une seule explication pour un problème précis. Grâce à l'approche par cas, on se réfère à un cas et la solution de ce cas est fournie.

B. Le raisonnement basé sur les règles

Le raisonnement par règle, le « *rules based reasonning* » vient d'un autre comportement typiquement humain lui aussi. Comme il a déjà été mentionné, le raisonnement est un processus cognitif permettant d'obtenir de nouveaux résultats, ou bien de faire la vérification d'un fait.

Le raisonnement par règle s'inscrit dans une logique mathématique, en utilisant la déduction, l'abduction ou l'induction.

Voici une représentation schématique [*Wiki01*] avec les notations classiques de la logique (→ pour l'implication) :

Déduction	Abduction
A a → b	B a → b
B	A

La règle d'abduction se lit ainsi:

1. si *b* est vrai
2. et si *a implique b*
3. alors A est vrai.

La règle de déduction se lit ainsi :

1. si *a* est vrai
2. et si *a implique b*
3. alors *b* est vrai.

Exemple : SI un animal à six pattes, ALORS c'est un insecte.

Il n'est possible de conserver la cohérence qu'avec la déduction, « si la théorie initiale est cohérente, alors toute théorie qui en est une conséquence déductive reste cohérente. » [3]

Appliquer une de ces 3 règles consiste à faire un raisonnement simple. Le raisonnement sera qualifié de déductif s'il s'appuie sur la déduction, et hypothétique s'il s'appuie sur l'abduction ou l'induction.

Un ordinateur résolvant des problèmes par application de règle est un « Rule base reasonning system » (CBR Systems) ; il est aussi doté d'une base de connaissance qui est constituée de l'ensemble des règles.

L'application de ces règles est faite par le moteur d'inférence, qui élabore la solution en choisissant les règles de production et leur séquence d'utilisation.

B.1 Le moteur d'inférence

Le moteur d'inférence [1] d'un système expert est en fait un compilateur de représentation interne. Il applique des règles que choisit un « *scheduleur* ». Un « *scheduleur* » cherche des règles applicables dans une situation ainsi que leurs priorités, et choisit une règle parmi toutes les candidates.

Le moteur d'inférence garantit aussi la cohérence, il écarte les incohérences, met à jour le contexte pendant l'inférence, c'est-à-dire qu'il remet à jour les règles applicables. Et enfin il garantit que les conclusions les plus plausibles soient retenues.

B.2 La limite du raisonnement par règle

Le « rule based reasonning » a une limite qu'il faut signaler.
Prenons par exemple [*Web04*] :
le programme " ticket " formé des règles suivantes :

- J'ai le ticket avec X si
- X me fait un sourire furtif et
- X maintient un contact oculaire prolongé

Dans le cas où le sourire et l'œillade sont là, est-ce vrai que j'ai le " ticket " ?
Car X pourrait être aveugle.
Il faut ajouter une nouvelle condition à la règle :

- et X non-aveugle

Sans parler de la définition des mots utilisés :

- Un sourire est furtif si sa durée est inférieure à une seconde
- Un sourire n'est pas un mouvement involontaire
- etc.

« En fait le « rule based reasonning » est approprié pour la résolution de tâches bien construites. » [4] Tout le champ des connaissances doit être exhaustif.

1.3.4 Sommaire

De plus en plus l'informatique assiste l'homme dans ses activités, dans un premier temps ses activités de gestion et dans un second temps ses activités de décision. Ainsi sont apparus les systèmes informationnels d'aide à la décision dans lesquels l'homme joue un grand rôle et ensuite les systèmes experts capables de résoudre par eux-mêmes des problèmes. On distingue deux types de systèmes experts : les systèmes experts basés sur le raisonnement par cas et les systèmes experts basés sur le raisonnement par règles.

1.4 Ré-ingénierie et restructuration

1.4.1 La restructuration

Le terme restructuration est vu comme une transformation code vers code, Chikofsky et Cross [5] nous donnent une définition générale : « La restructuration est la transformation d'une forme de représentation vers une autre au même niveau d'abstraction, tout en préservant le comportement externe du système. »

Mais la restructuration peut être vue aussi sous d'autres angles. Elle peut être le passage d'un système non structuré vers un système structuré, en connaissant la structure mais pas le contenu ni même le but du système. Par exemple, il est possible de changer une imbrication d'instruction conditionnelle en un « case » ou le contraire, sans pour autant connaître le domaine ou le but du problème.

La restructuration peut également être une forme de maintenance de prévention pour améliorer l'état physique d'un système en prenant de nouveaux standards.

Enfin la restructuration est aussi un ajustement du système pour répondre par la suite à de nouvelles contraintes ne nécessitant pas de réévaluation d'un niveau d'abstraction plus élevé.

1.4.2 Ré-ingénierie

L'essence même de la ré-ingénierie est de refaire radicalement la conception des processus d'une organisation. Lors d'un changement, au lieu de réviser chacune des fonctionnalités d'une firme, la re-ingénierie propose de repenser le processus complet. C'est une série de nouvelles fonctionnalités qui verront le jour.

La ré-ingénierie au fil du temps a été mal vue car elle s'assimilait aux licenciements massifs.

Mais cet amalgame est faux, ce sont les compagnies qui ont appelé ré-ingénierie, leurs réductions de personnel.

Les grands partisans de la ré-ingénierie sont Michael Hammer et James Champy. Dans leurs livres, tels que *Reengineering the Corporation*, *Reengineering Management*, et *The Agenda*, ils expliquent que beaucoup trop de temps est gaspillé à la révision.

La ré-ingénierie est un processus délicat, il faut avant toutes choses que ce processus aboutisse. Pour cela il faut l'accord des personnes qui savent comment tourne une organisation. C'est un processus souvent tentant… quel dirigeant n'aurait pas l'idée de prendre quelque chose de défectueux et de le concevoir sous une autre forme. Mais la ré-ingénierie ne doit pas viser uniquement ce qui doit être sans se soucier du pourquoi c'est présentement ainsi. Il ne faut donc pas être réticent envers un processus de ré-ingénierie mais être prêt, savoir ce qui doit être oublié et ce qui sera réinventé, ainsi qu'avoir l'accord des personnes concernées.

La ré-ingénierie comme nous l'avons vu s'applique aux organisations, mais elle peut aussi s'appliquer à beaucoup d'autres domaines comme l'informatique, avec les mêmes problèmes et exigences. Chikofski et Cross [5] donnent une définition plus adaptée à l'informatique de la re-ingénierie : « la re-ingénierie est l'examen et l'altération d'un système pour le reconstituer dans une nouvelle forme et en implémenter cette nouvelle forme. »

1.4.3 Sommaire

Nous avons deux processus bien distincts qui permettent la remise à niveau : la ré-ingénierie et la restructuration. Le premier consiste à examiner un système et le refaçonner entièrement, avec une possibilité d'implication de changement de comportement de ce système, tandis que le second ne change pas le comportement.

Chapitre 2 La maintenance de logiciel

Comme nous le savons le logiciel *SM*^{Xpert} a pour domaine d'application la maintenance de logiciel. Nous allons donc voir les connaissances de base en maintenance de logiciel.

2.1 Introduction

Tout d'abord, un logiciel commence par son développement, ensuite une fois fini et donnant satisfaction aux exigences du client, il devra évoluer pour satisfaire les besoins d'un environnement en progression. De plus, certaines anomalies peuvent être découvertes lors de son l'utilisation, un processus de maintenance est donc nécessaire. Mais cela ne signifie pas que le cycle de vie de la maintenance débute lors de sa mise en production. «Le cycle de vie de la maintenance débute bien avant par une participation active des responsables de la maintenance tout au long du processus de développement. » [11]

Il existe de multiples définitions de la maintenance, elles ont été émises de manière successive dans l'histoire.

Ainsi, en 1983, Martin & McLure dans *Software Maintenance : The Problem and its Solutions* donnaient cette définition : « Changements qui doivent être effectués sur un logiciel après sa livraison à l'utilisateur. » Quelques années plus tard, la définition s'est quelque peu affinée pour donner : « Modification d'un logiciel, après sa livraison, afin de corriger des défaillances, améliorer sa performance ou d'autres attributs ou de l'adapter suite à des changements d'environnements. » *IEEE 610.12-1990 : IEEE Standard Glossary of Software Engineering Terminology.*

Finalement, la dernière définition en date est encore plus détaillée. Elle est donnée par le SWEBOK (Software Engineering Body of Knowledge) en 2004 : « la totalité des activités qui sont requises afin de procurer un support, au meilleur coût possible, d'un logiciel. Certaines activités débutent avant la livraison du logiciel, donc pendant sa conception initiale, mais la majorité des activités ont lieu après sa livraison finale (l'équipe de développement ayant maintenant terminé son travail et étant affectée à d'autres travaux). »

On dénote une certaine évolution dans les différentes définitions. La première définition est assez allusive, ne précisant même pas en quoi consiste une modification. La deuxième définition précise, quant à elle, ce que signifient les modifications et ce qui les provoque. Indirectement on sait aussi que la maintenance, ne se réalisant qu'une seule fois, n'est pas une activité temporelle.

Enfin, nous pouvons remarquer que dans la dernière définition, des critères comme l'optimisation entrent en compte, le début des activités de la maintenance est redéfini. On voit donc, par le biais de ces définitions, que progressivement les connaissances en maintenance se sont étoffées.

La maintenance de logiciel a pris au fil du temps de plus en plus d'importance, elle ne peut plus se réaliser avec négligence ; il faut une organisation pour mener à bien des activités de maintenance. Ce chapitre présente l'ensemble des connaissances nécessaires à la maintenance de logiciel.

2.2 Différence entre développement et maintenance de logiciel

Le développement de logiciel possède des similitudes avec la maintenance, ce sont deux activités parfois difficiles à distinguer, surtout lorsque les développeurs de logiciel effectuent eux-mêmes la maintenance de ce dernier. Il y a en effet beaucoup d'activités communes entre le développement et la maintenance de logiciel (analyse, conception, codage, gestion de la configuration, essais, revues et documentation). Mais pour chacune de ces activités, le contexte est différent. Pour la maintenance, ces activités sont faites dans une optique d'amélioration ou de correction avec un délai souvent plus bref et des ressources humaines restreintes.

Les outils utilisés pour le développement peuvent également servir à la maintenance, mais à nouveau leur contexte est différent.

Un ingénieur de la maintenance doit donc avoir les qualités requises d'un développeur en plus des aptitudes requises pour les activités propres à la maintenance. Car celle-ci possède bien évidement ses activités propres. En voici un résumé :

1. L'équipe de maintenance doit pouvoir gérer les événements et requêtes de modifications venant de la clientèle, tout en continuant ses travaux. Il faut donc un processus d'acceptation ou de rejet du travail. L'arrivée de ces événements est aléatoire, alors que pour le développement tout est prévu sur un plus long terme, planifié avec un échéancier, et aboutit sur une livraison du produit. Il est évidemment impossible, vu le caractère aléatoire des demandes, de planifier tout dans un processus budgétisé comme pour le développement ; de plus « les travaux sont ordonnés de manière à satisfaire l'utilisateur, à court terme, et s'assurer du bon fonctionnement quotidien des logiciels en opérations. » [11]
2. Il faut évaluer les requêtes de demande et les classer par ordre de priorité. Il faut estimer l'effort requis pour modifier le logiciel existant. « Si l'effort estimé est peu élevé et peut être accommodé à l'intérieur des ressources et disponibilités de l'équipe de maintenance pour gérer le quotidien de la maintenance, la requête est alors exécutée à l'intérieur de la gestion de la file d'attente. » [11] Si l'effort estimé est élevé par rapport au temps limite adjugé par une organisation ou par rapport à sa marge de manœuvre budgétaire, alors la requête devient un nouveau projet avec un budget spécifique, une équipe et un échéancier. Il faut aussi préciser que les priorités peuvent changer rapidement et à n'importe quel moment suivant les requêtes et prendre aussi priorité sur un travail en cours.
3. Comme signalé, les requêtes de modifications arrivent de manière aléatoire, la charge de travail de la maintenance est donc gérée par « une technique de file d'attente, souvent supportée par un logiciel de bureau d'aide '*help desk*'. »

« Une dernière grosse différence entre le développement et la maintenance est que le développement est « *requirement-driven* », alors que la maintenance est « *event-driven* », ce qui signifie que le stimuli qui initie la maintenance est un événement imprévisible. » [12]

2.3 Qui fait la maintenance

Il existe deux modèles distincts en entreprise. Dans le premier modèle organisationnel, la maintenance est effectuée par les développeurs eux-mêmes. Dans le second modèle, c'est une organisation de maintenance, indépendante des développeurs, qui effectue la maintenance.

Mais dans le premier modèle organisationnel, souvent les développeurs n'aiment pas faire de la maintenance et préfèrent travailler sur le développement de nouveaux projets.

On soupçonne plus facilement un manque de transparence des organisations effectuant leur propre maintenance de logiciel, ainsi les logiciels livrés seraient de moins bonne qualité et peu documentés afin qu'il soit plus difficile pour une autre équipe de maintenir le logiciel. Néanmoins cet inconvénient permet « une meilleure documentation des logiciels, un processus plus formel et contrôlé de transition du logiciel développé, une plus grande spécialisation du programmeur de maintenance, une meilleure gestion des requêtes de changements et une plus grande satisfaction des employés. » [11]

2.4 Le processus de la maintenance du logiciel

Au tout début de l'industrie du logiciel, la maintenance et le développement n'étaient pas distincts. Cela a donné une vue du cycle de vie du développement dans laquelle la maintenance était considérée comme une étape finale du cycle. La maintenance a été longtemps vue sous cet angle et parfois même complètement ignorée, alors que pourtant les premiers modèles de cycle de vie pour le processus de maintenance sont apparus en réalité très tôt dans l'industrie du logiciel.

Ils étaient « en général coupés en 3 phases.

1) Compréhension,
2) modification et
3) validation du changement au logiciel. » [11].

Ces cycles ont été améliorés ; ces 15 dernières années des consensus internationaux ont défini la terminologie et les cadres de références pertinents à la maintenance actuellement utilisés.

Voici un diagramme général du cycle de vie de la maintenance. Bien évidemment chaque organisation peut faire son propre cycle de vie, mais dans l'ensemble il devrait être proche de celui proposé.

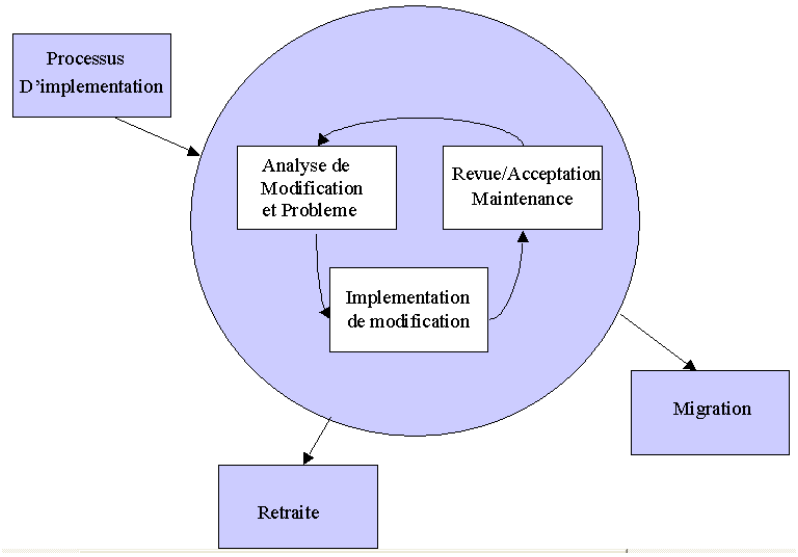


Figure 11 : Meta processus de la maintenance du logiciel [11]

Voici plus en détail les différents processus :

- Le processus de l'analyse des modifications et problèmes. Ce processus commence lors de la prise en charge du logiciel par le groupe de maintenance. Il faut y analyser chaque requête (exemple : modification), c'est-à-dire vérifier leur validité et documenter la requête, ensuite chercher une solution et la documenter pour finalement obtenir une autorisation d'effectuer les modifications.
- « Le processus de l'implantation d'une modification autorisée. On s'attarde ici à mettre en œuvre les processus similaires à ceux du développement. » [11]
- Le processus de revue/acceptation d'une modification : c'est une validation de la solution en la soumettant à la personne ayant émis la requête.
- Le processus de migration : c'est un processus d'exception, il entre en jeu lorsque le logiciel doit par exemple changer de plate-forme sans pour autant subir de modifications fonctionnelles.
- Le processus de retrait : c'est également un processus exceptionnel qui consiste à « enlever un logiciel de l'environnement opérationnel. » [11]
- Le processus d'implémentation est plus externe aux processus opérationnels de la maintenance. Ce sont les activités de préparation et de transition du logiciel. C'est-à-dire par exemple : création du plan de maintenance, préparation de la gestion des problèmes identifiés pendant le développement, etc.

2.5 Les Catégories de travaux de la Maintenance du Logiciel

La maintenance vient d'un événement pouvant être un rapport de problème ou bien une demande de changement, c'est à partir de cela qu'il est possible de définir l'activité de maintenance nécessaire [12].

La maintenance se divise en 4 catégories d'activités : corrective, adaptative, perfective et préventive. Ces activités ont été catégorisées suivant deux critères :

Critère 1: le travail consiste soit en une correction, soit en une amélioration au logiciel.
Critère 2: le travail est soit proactif, soit réactif.

	Correction	Amélioration
Proactive	Préventive	Perfective
Réactive	Corrective	Adaptative

Catégorie de maintenance [11]

Correction : « corriger une défection. Exemple : une anomalie entre le comportement attendu d'un produit et le comportement observé. » [12]

La correction préventive est un changement de l'implémentation sans changer les exigences.

Amélioration : « implémentation d'un changement du système qui change son comportement. » [12] L'amélioration adaptative ajoute de nouvelles exigences, tandis que l'amélioration perfective est un changement des exigences existantes.

Il est important de savoir distinguer si on fait de la maintenance corrective ou perfective, car il se peut qu'un problème rapporté par un utilisateur soit la demande d'une fonctionnalité nouvelle qui à l'origine n'était pas requise et dans ce cas le rapport de problème mène à une amélioration.

Les mainteneurs ont besoin de comprendre le produit et d'avoir différents outils. La maintenance corrective pourrait se faire en étant seulement capable de trouver et localiser tous les changements à effectuer dans le code. Tandis que pour la maintenance perfective, il faut une large compréhension du produit. Ceci est très possible avec une documentation de qualité,

mais également avec des outils, comme les outils de mesure de complexité cyclomatique permettant d'avoir une idée de la complexité d'un logiciel, ou alors localiser les endroits sensibles du code. C'est une information importante, par exemple pour la gestion de la priorité.

Nous verrons dans les problèmes liés à la maintenance que la documentation joue un rôle important et nous verrons aussi dans la définition de l'ontologie de la maintenance que la documentation est un facteur ayant également un rôle.

2.6 Ontologie de la maintenance de logiciel

Après avoir vu les connaissances fondamentales de la maintenance, il serait intéressant d'en avoir une description plus riche encore.

Pour cela, une ontologie de la maintenance sera présentée. « L'ontologie est une spécification de la conceptualisation. » [12] C'est par le biais d'une identification des facteurs influençant la maintenance que cela sera réalisé.

La Figure 12 montre les facteurs qui influencent le processus de maintenance et leur classification.

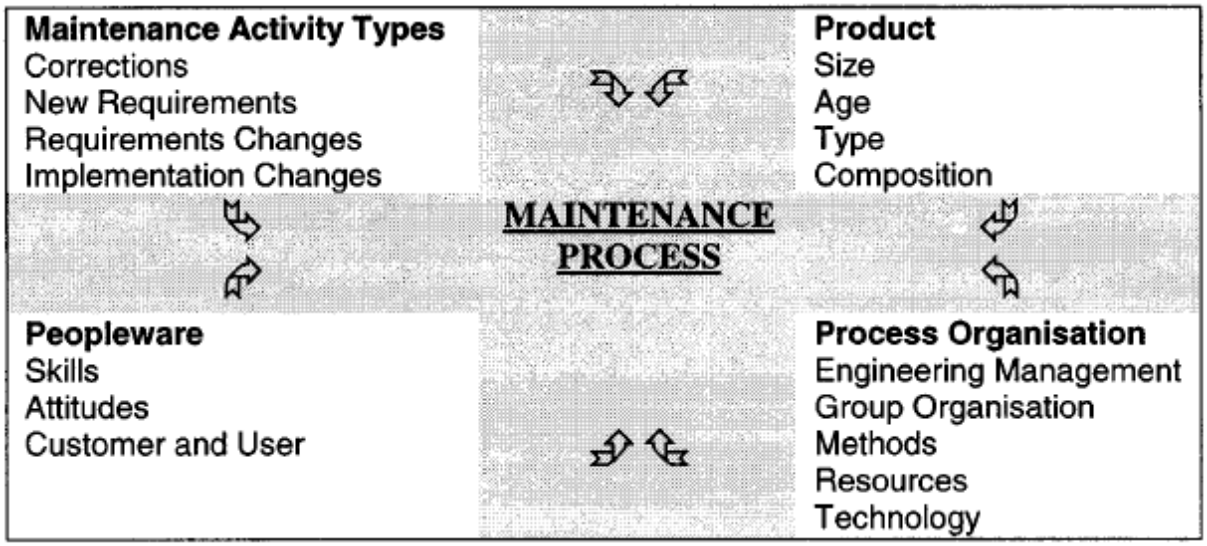


Figure 12 : Vue d'ensemble du domaine des facteurs influençant la maintenance de logiciel [12]

A présent il faut décrire les facteurs ainsi que leur caractéristiques ayant un impact sur l'activité de la maintenance.

2.6.1 Le produit

Voici l'ontologie du produit de la maintenance :

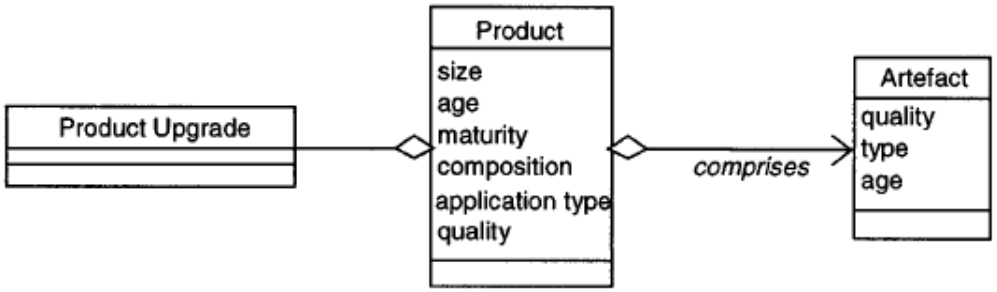


Figure 13 : ontologie du produit en maintenance [12]

Le produit	Logiciel ou package à modifier.
Mise à jour du produit	Nouvelle version du produit, un nouveau patch, etc.
Artefact	Le code, mais aussi les documents relatifs au code. Ces documents sont le design, la spécification des exigences, etc.

Définition de l'ontologie du produit en maintenance

La taille du Produit

Le nombre et l'organisation d'une équipe varieront en fonction de la taille du produit.

Par exemple : plus la taille du produit sera grande, plus il sera susceptible que tous les membres d'une même équipe ne comprennent pas le produit de la même façon. Ce qui rend difficile le diagnostic des problèmes et l'identification des modifications à réaliser, car il y a plus de probabilité de malentendu. Ainsi l'activité de la maintenance sur de gros produits peut être moins productive que sur de petits produits.

Une personne est souvent affectée à la maintenance d'un petit produit, alors qu'il y aura déjà une équipe pour un produit de taille moyenne et plusieurs équipes pour un produit de grande taille.

Type ou domaine d'application

Il a été observé des différences majeures de productivité entre produits provenant de domaines variés. Par exemple « la maintenance d'un système de sécurité doit à tout prix préserver la fiabilité du système. » [12] Ainsi le domaine d'application peut imposer des contraintes faisant varier la productivité de la maintenance.

Age du produit

Un vieux produit avec une vieille technologie de développement sera difficile à maintenir. Il y a avant tout un gros risque qu'il soit difficile de trouver quelqu'un ayant encore les connaissances des technologies désuètes. Il sera aussi difficile de retrouver les concepteurs de ce produit, ainsi que la documentation. Il se peut donc qu'il y ait des parties du logiciel que personne ne puisse comprendre.

On présume donc qu'en général la maintenance sera plus performante pour des produits récents que pour de vieux produits.

Maturité du produit

La maturité ne doit pas être confondue avec l'âge. C'est le cycle de vie du logiciel après sa sortie. L'activité de maintenance change suivant l'étape du cycle de vie dans laquelle se trouve le produit.

Etape du cycle de vie	Tâche la plus courante	Nombre d'utilisateurs
Stade de l'enfance : arrivées des rapports d'erreurs par les utilisateurs initiaux, après livraison du logiciel.	correction	Petit
Stade de l'adolescence : le nombre d'utilisateurs augmente, les rapports d'erreurs sont prédominants, et il commence à y avoir des demandes de modifications du comportement du système.	Corrections et changement des exigences	En croissance
Âge adulte : il n'y a presque plus d'erreur dans le produit. Les demandes de nouvelles fonctionnalités sont prédominantes, et si elles sont acceptées par un grand nombre d'utilisateur, elles seront faites. Si les modifications s'enchaînent, alors il y aura besoin de restructurer le code pour éviter un design dépréciant.	Nouvelles exigences et implémentation de ces exigences	Maximum
Sénile : il existe des nouveaux produits disponibles et il reste seulement peu d'utilisateur à supporter. Ordinairement il ne reste que des corrections à fournir.	Correction	Décroissant

Cycle de vie du produit après livraison

La composition du produit

La composition d'un produit affecte également la maintenance et plus particulièrement les aptitudes en maintenance ainsi que les outils nécessaires. « Si le produit est généré du design, alors les ingénieurs de la maintenance doivent avoir accès aux outils de génération de code. »

Qualité du produit

La qualité fournie par le processus de développement d'un logiciel ajoute des contraintes au processus de maintenance. Il est évidemment plus facile de maintenir des produits de bonne qualité, sachant que « qualité » ici signifie : documentation, structure du produit…

Ce facteur est encore plus important lorsqu'il n'y a pas moyen d'entrer en contact avec l'équipe ayant développé le produit.

Il faut remarque que l'existence d'une documentation n'implique pas que le produit soit pour autant de qualité, il faut également que cette documentation soit de qualité. B. A. Kitchenham et Al définissent une documentation de qualité comme une documentation complète, précise et lisible.

2.6.2 Les activités de maintenance

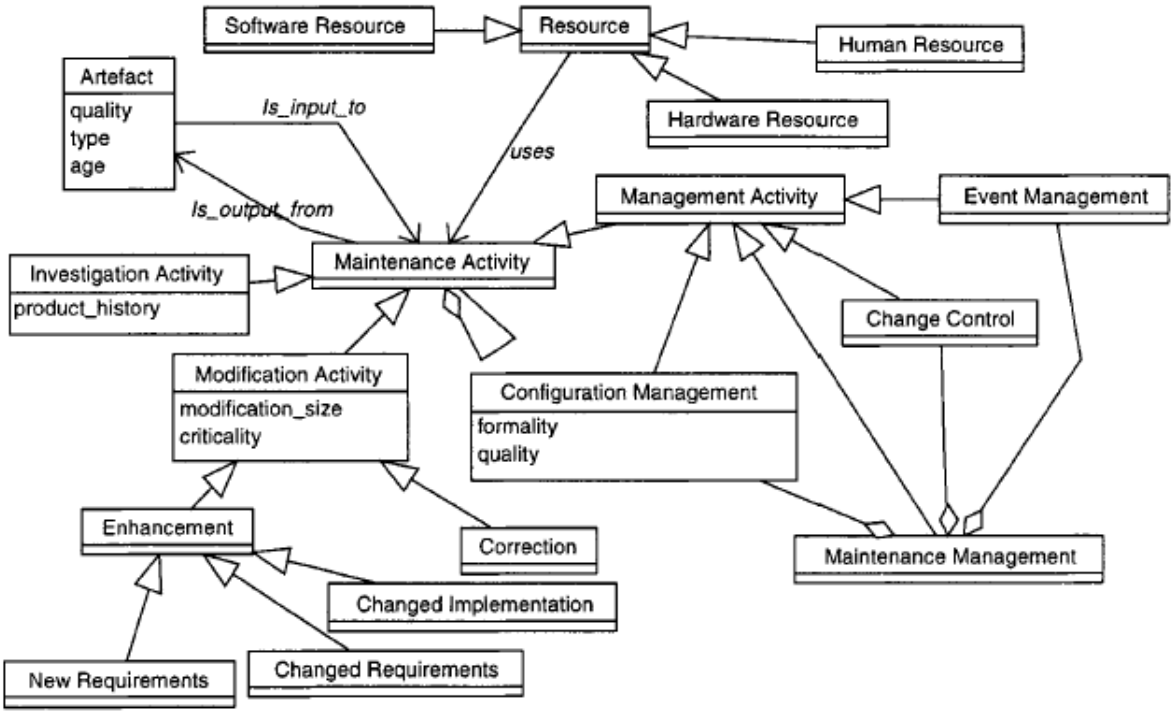


Figure 14 : ontologie de l'activité de maintenance [12]

Activité d'investigation	Évaluer l'impact d'une modification.
Activité de modification	(cf. : catégories de travaux de la maintenance de logiciel).
Activité de gestion	Activités de gestion des requêtes, activités permettant de s'assurer de l'intégrité d'un produit en maintenance et de ses différentes versions.
Ressource	Ressources humaine, matérielle et logicielle permettant une activité de maintenance.

Définition de l'ontologie de l'activité de maintenance

La maintenance commence soit à cause des utilisateurs, soit à cause des mainteneurs. Les causes peuvent être un rapport d'erreur ou une demande de changement. Ces causes seront étudiées par un ingénieur en maintenance, dans l'étape de l'*analyse des modifications et problèmes* du processus de maintenance, afin de déterminer si des modifications de maintenance sont nécessaires ou pas.

Les catégories maintenance ont précédemment été présentées. Il est utile de rappeler rapidement les deux activités principales sans pour autant s'attarder sur leurs variantes :

- Correction : « corriger une défection. Exemple : une anomalie entre le comportement attendu d'un produit et le comportement observé. » [12]
- Amélioration : « implémentation d'un changement du système qui change son comportement. » [12]

Importance d'une amélioration/correction

Une caractéristique des activités de la maintenance affectant la productivité et l'efficacité est l'importance de l'activité de maintenance. Une large amélioration/correction demandera un plus gros effort et plus d'ingénieurs en maintenance. Cet effort sera amplifié suivant l'importance du produit.

Connaissance du produit

Le degré de compréhension et les outils à disposition sont aussi des facteurs. Lors d'une maintenance corrective, les aptitudes nécessaires peuvent se réduire à pouvoir corriger des fautes de codes et ainsi procéder seulement à des changements locaux. Mais lorsqu'il s'agit d'une amélioration, il faut évidemment connaître le produit.

2.6.3 L'environnement humain (peopleware)

Le développement de logiciel, tout comme sa maintenance, demande beaucoup de ressources humaines travaillant en groupes. Ainsi le facteur humain et social ne peut pas être ignoré en tant que facteur affectant la maintenance. Nous allons donc voir l'ontologie de l'environnement humain. Il n'y a pas de définition de chaque élément de l'ontologie car leurs noms sont suffisamment parlants.

Dans le processus de maintenance, il y a deux types d'équipe :

- l'équipe s'occupant de la maintenance (« *engineers* » sur la Figure 14) ;
- l'équipe s'occupant du client (« *managers* » sur la Figure 14).

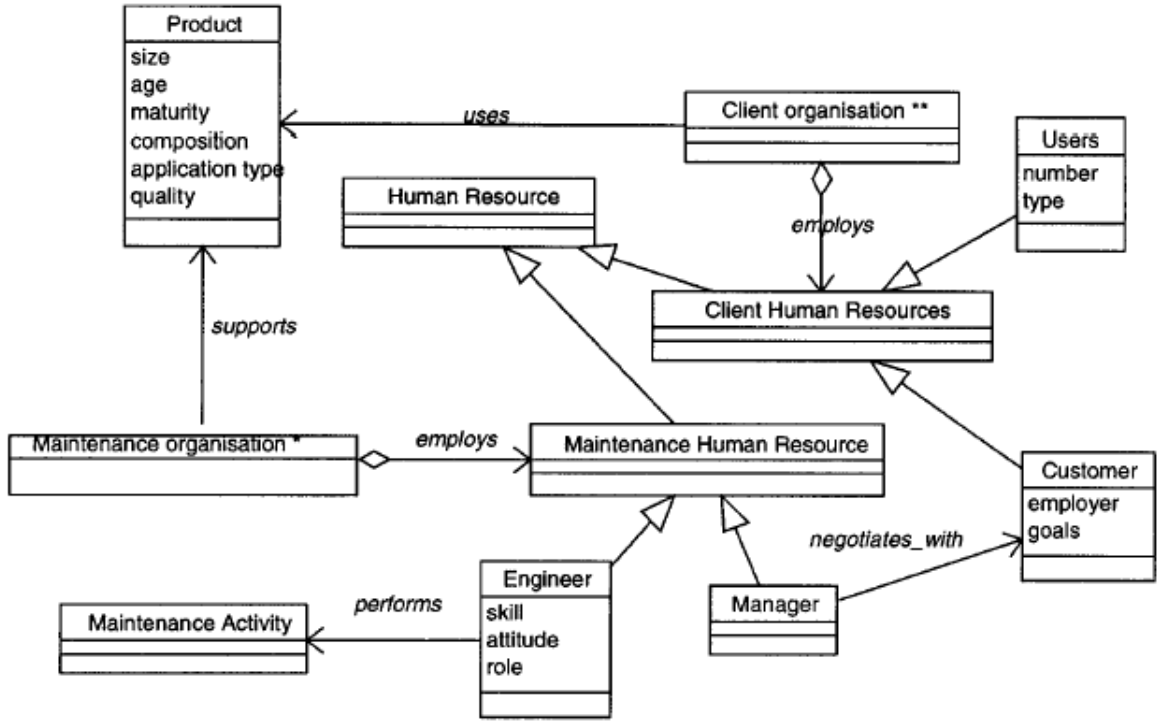


Figure 15 : ontologie de l'environnement humain [12]

L’équipe s’occupant de la maintenance

La motivation

Les motivations et les attitudes du personnel influent sur leur travail. Les gens ont une perception négative de la maintenance. Ceci sera expliqué plus en détail dans la section intitulée : problème de la maintenance du logiciel. Disons seulement que cela affecte directement la performance de la maintenance.

Séparation entre développeur et ingénieur de la maintenance

Le fait d’avoir ou non une séparation stricte entre équipe responsable du développement et équipe responsable de la maintenance est important. Certaines organisations n’ont pas cette séparation. Cela dépend du produit développé. Le produit est en continuelle évolution et de nouvelles versions sortent régulièrement. Les développeurs font eux-mêmes les corrections et les améliorations sont inclues dans un processus continu. Ainsi les outils permettant le développement et ceux permettant la maintenance ne sont pas différents.

Dans le cas contraire, celui où il y a nettement une séparation, voire même deux compagnies différentes, une de développement et une de maintenance, il est clair que les mainteneurs auront besoin d’outils spécialisés (des outils de rétro ingénierie par exemple).

L’équipe s’occupant des clients

Diversité

La diversité de la clientèle qui agrandit les possibilités de tâches de maintenance. Plus elle sera grande, plus il y aura différents types de problèmes.

Financement

Le financement provient des clients qui éprouvent la volonté d’améliorer un logiciel ; il va de soit que la maintenance dépend de leur satisfaction. En effet, si les exigences ne sont pas atteintes au cas où elles seraient mal comprises, les clients ne seront pas satisfaits.

2.6.4 Le processus de maintenance

Il faut faire une distinction entre le processus de maintenance réalisé par les ingénieurs en maintenance pour une modification spécifique et le processus d’organisation de la gestion des requêtes de maintenance des clients et des mainteneurs eux-mêmes.

Processus pour une modification spécifique

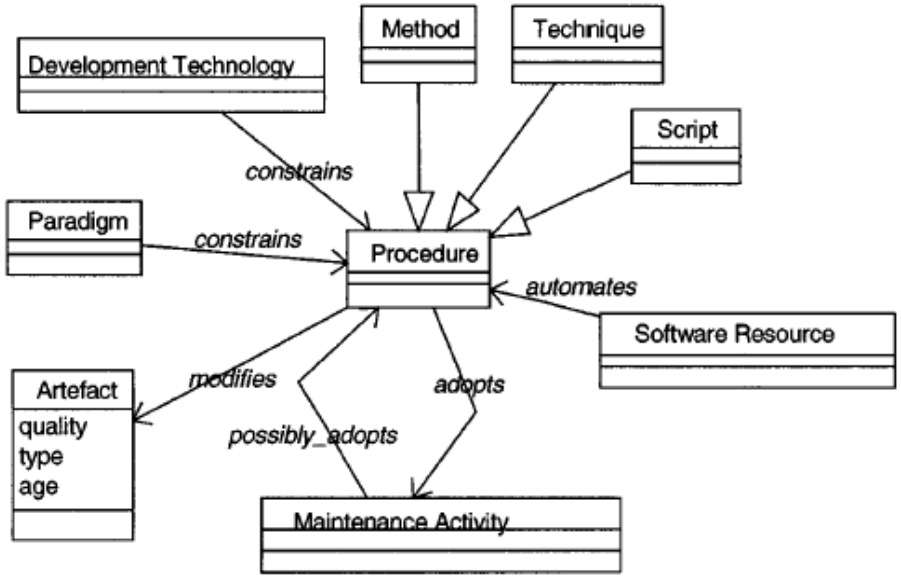


Figure 16 : ontologie d'une modification spécifique [12]

Paradigme	La philosophie adoptée durant le développement initial du produit.
Procédure	La procédure même d'une activité de maintenance. Cela peut être une méthode, une technique ou un manuscrit (« *script* »). Certaines procédures peuvent être choisies pour réaliser une activité spécifique.
Méthode	Une procédure systématique définissant les pas et les heuristiques afin de réaliser une activité de maintenance.
Manuscrit technique	Une procédure utilisée pour accomplir une activité, mais étant moins rigoureusement définie qu'une méthode.

Paradigme de développement

Le paradigme de développement d'un logiciel affecte la performance de la maintenance, plus précisément la contraint. Les mainteneurs ont besoin des aptitudes nécessaires à la compréhension du langage utilisé.

Technologies de développement

Les technologies permettant le développement d'un logiciel présentent un risque. Il se peut qu'elles deviennent un jour inutilisables. Il faut donc s'assurer de ce risque pour les logiciels ayant un long cycle de vie.

Enfin, le degré d'automatisation des procédures influe aussi sur la performance.

L’organisation de la maintenance

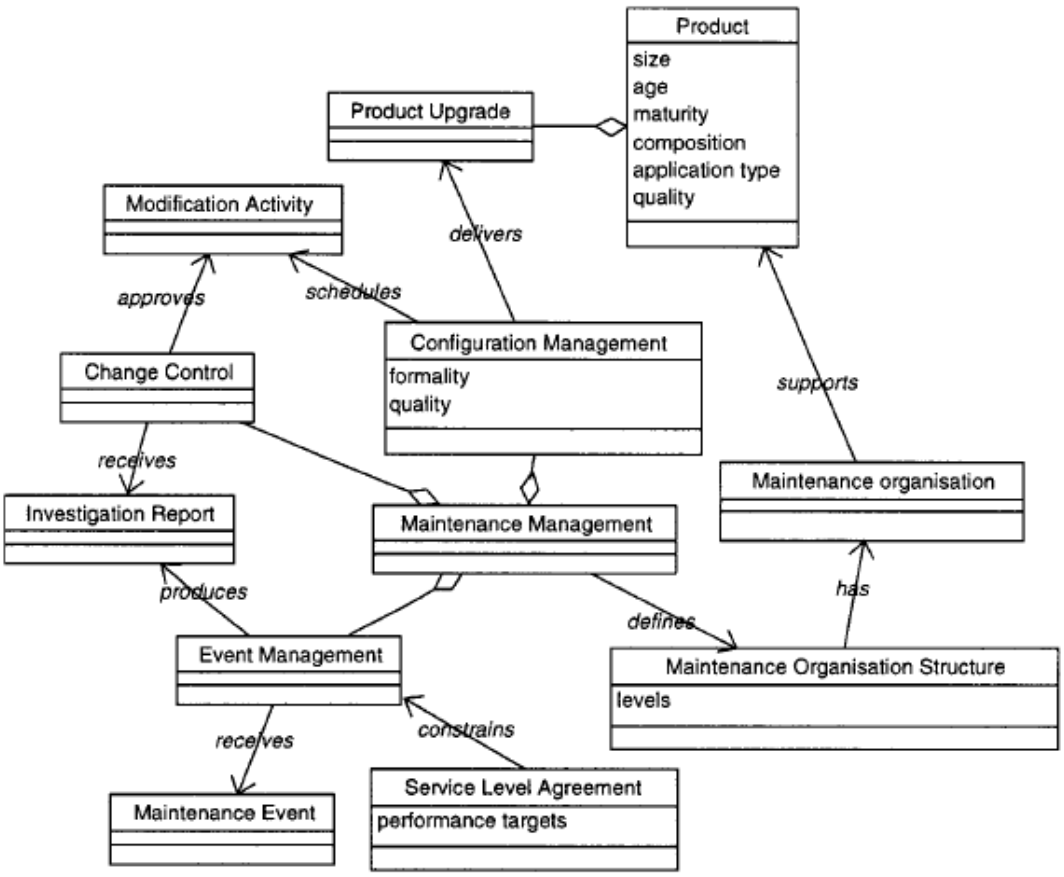


Figure 17 : ontologie de l’organisation de la maintenance [12]

« Service Level Aggrement » (SLA)	Accord entre le fournisseur de service de maintenance et le client qui spécifie ses attentes de service.
La gestion des configurations	C’est la gestion des versions d’un produit, et de son état de modification.
Le contrôle de changement	Le processus responsable d’évaluer les résultats d’une investigation de maintenance et la décision d’un changement probable.
Evénement	Un rapport de problème ou une requête de changement.

Requêtes

L’influence de la quantité de requêtes de maintenance sur les performances et son organisation n’est pas négligeable. Exemple : trop de demandes impliquerait un manque de ressources humaines.

Outils

Il est important d’avoir des outils de support à la maintenance. La plupart des organisations ont leurs outils de contrôle de configuration et il existe aussi beaucoup d’outils de support à la gestion des événements permettant de récupérer les demandes et de les tracer.

2.7 Problèmes de la maintenance du Logiciel

Le problème majeur en maintenance de logiciel n'est pas de nature technique, c'est sa gestion qui est délicate.

On distingue deux catégories de problèmes de la maintenance :

1. Les problèmes internes venant de la perception des employés et gestionnaires de la maintenance de logiciel.
2. Les problèmes externes venant de la perception de la clientèle.

2.7.1 Problèmes internes

Plus un logiciel fait objet de maintenance, plus la complexité de sa structure et sa taille s'accroissent et progressivement s'élève l'effort de maintenance. Ceci est un problème technique fréquent car les logiciels sont de plus en plus susceptibles de faire l'objet de maintenance, et ce plusieurs fois ; mais comme il a déjà été précisé, les problèmes techniques ne sont pas les plus importants.

Le caractère aléatoire des requêtes et la difficulté de les comprendre ; l'obligation de réagir rapidement aux pannes et de s'assurer que le service est promptement et bien restauré, c'est-à-dire respecter un niveau de service établi afin de garder la confiance de la clientèle ; la gestion de priorités changeantes, car les organisations des utilisateurs changent vite : tout cela crée une très mauvaise et stressante atmosphère pour une organisation de maintenance.

A cette atmosphère vient s'ajouter une condition à laquelle on ne pense pas forcément : la motivation. En effet « en maintenance la motivation peut poser problème car l'activité de maintenance peut être perçue comme moins importante que le développement. » [12] Il est très difficile de mesurer la contribution de la maintenance, ainsi que sa performance, il y a par conséquent un manque de considération et de respect de la maintenance. Les novices y sont le plus souvent assignés et elle est aussi perçue comme une punition. Les mainteneurs, conscients du manque de considération, se voient aussi contraints « d'accepter, avec résignation, le prochain logiciel développé, peu importe son niveau de qualité. » [11] Ils sont donc aussi contraints de résoudre des problèmes qu'ils n'étaient pas censés régler. Les ingénieurs de la maintenance ont ainsi l'impression d'être les seuls responsables du bon fonctionnement, de la gestion des logiciels et du support à la clientèle.

Les ingénieurs de la maintenance ont également peu de support technique et d'outils spécifiques à la maintenance et doivent parfois faire face à une documentation du logiciel incomplète ou absente. Cela arrive souvent lorsque le budget pour le développement n'est pas suffisant ; dans ce cas les étapes d'analyses sont moins prises en compte. Mais ce souci d'économie est une idée fausse car la maintenance sera plus onéreuse. Le problème de documentation vient directement du processus de développement, on voit donc qu'un processus de développement mature garantit une certaine facilité de maintenance, non pas spécialement par la documentation mais également par la qualité du code.

Il est aussi très difficile de retracer la création d'un logiciel et de ses changements. En outre, les changements ne sont que très rarement documentés.

Il y a aussi des problèmes indirectement liés aux problèmes internes de la maintenance, comme par exemple le fait que la maintenance n'est pas enseignée comme elle est vécue en industrie. « Il en résulte un manque de connaissances et techniques de la maintenance pour les candidats potentiels pour des emplois en maintenance du logiciel. » [11]

Un dernier phénomène non négligeable est la culture. Certaines cultures s'articulent davantage sur la qualité et ont l'idée que l'amélioration augmente régulièrement la satisfaction du client. Ces cultures donnent donc une place plus importante à la maintenance.

2.7.2 Problèmes externes

Un premier problème est que la grande majorité des coûts dans le cycle de vie des logiciels semble liée à la maintenance. A. April [11] rapporte que les études faites dans ce sens appuieraient cette perception des utilisateurs.

La maintenance est un travail laborieux et les salaires des programmeurs prennent une grosse partie des coûts, mais ces coûts sont des coûts justifiés. Ce qui rend la maintenance si chère à la vue des utilisateurs provient d'autres problèmes.

Le caractère aléatoire des multiples requêtes des utilisateurs doit pouvoir être géré par des mécanismes de gestion de file d'attente. Car il arrive souvent que les utilisateurs ne précisent pas leurs priorités ou alors les changent eux-mêmes constamment, et il n'est pas rare que certaines de leurs priorités soient oubliées ou même qu'une tâche de maintenance en cours soit postposée. Il se peut également que des pannes surviennent. Ces problèmes sont à nouveau bel et bien des problèmes de gestion. Et cela peut devenir frustrant pour un utilisateur d'attendre à tel point que ce dernier finisse par proposer ses propres solutions ou même sous-traite chez un externe la maintenance.

Mais cette vision de lenteur est-elle entièrement bien fondée ? A. April [11] reprend une étude de Lientz et Swanson dans *Software Maintenance Management*, qui, sur base d'un sondage auprès de 487 groupes de maintenance, a rapporté que 55% des requêtes des utilisateurs étaient en réalité des nouvelles exigences. Mais ce n'est pas tout, les requêtes des utilisateurs ne constituent pas la seule source de requête. Il y a aussi les requêtes des opérateurs, des gestionnaires de projets et des études de re-ingénieries qui, elles aussi, viennent chambouler les priorités.

La clientèle n'est pas consciente que les services de maintenance font en réalité plus que de la maintenance. Une fois encore, une maintenance mieux gérée éviterait cette confusion.

Un dernier point à souligner sur cette vision de lenteur de service, c'est qu'elle est aussi due à une incompréhension de la différence entre matériel et logiciel. Le matériel est palpable, il est facile de détecter un composant défectueux et de le remplacer, sans pour autant changer les autres composants. Ce n'est pas encore le cas pour la maintenance de logiciel.

2.8 La mesure pour la maintenance

Maintenant que nous avons vu maints problèmes en maintenance, il serait intéressant de savoir comment les contrôler. Pour pouvoir contrôler, il faut pouvoir prédire, comparer, évaluer. Les outils de mesure sont utiles dans ces activités. En ce qui concerne la maintenance, c'est prédire le comportement de ressources, c'est mesurer le processus de maintenance et son produit, ainsi qu'évaluer le service ou le comparer.

2.8.1 L'estimation des ressources

L'estimation des ressources (nombre d'employés, budgets, etc.) se fait le plus souvent suivant deux approches. La première est basée sur l'expérience, la seconde utilise des modèles tels que COCOMO-maintenance.

2.8.2 La mesure du processus de la maintenance

L'objectif ici est de mettre sous contrôle le processus de la maintenance.

Bien que la mesure de la maintenance ressemble fortement à la mesure du développement, ces deux mesures doivent être distinctes car leurs exigences sont différentes. Voici plusieurs exemples de mesure du processus de la maintenance.

Le premier exemple est un ratio simple, le *'Temps moyen de changement'* (représentant le temps moyen pour effectuer un changement sur un logiciel après son développement) est une autre perspective intéressante à mesurer. Une interprétation possible de ce ratio est que son importance est un indice de la 'maintenabilité' d'un logiciel. Si le temps moyen de changement d'un logiciel est court, alors c'est qu'il est facilement maintenable.

Il existe aussi une mesure du coût de la 'maintenabilité' du logiciel (coût de changement sur un logiciel après son développement). En utilisant le ratio du nombre de défaillances en relation avec le coût du projet initial de développement. On peut ainsi savoir quel processus de développement offre un coût de maintenance plus faible.

2.8.3 La mesure du produit de la maintenance

Il existe peu de mesures spécifiques à la maintenance pour mesurer son produit ; mais il existe beaucoup de mesures du code source, mesures pour le développement, donnant des informations sur la 'maintenabilité' d'un logiciel.

Comme il a déjà été précisé, la qualité du logiciel a un impact sur la maintenance. Il est important d'avoir un logiciel de qualité. Il existe des mesures de la qualité d'un logiciel qui prennent en compte l'aspect 'maintenabilité'.

Une mesure possible pour le produit est l'effort nécessaire pour analyser et effectuer des modifications sur un logiciel ou mesurer les attributs du logiciel qui vont avoir un impact sur l'effort de maintenance.

Les mesures des attributs d'un logiciel viennent en général du code. Il est possible de mesurer de différentes façons la complexité de la structure en étudiant une représentation graphique du code source. L'exemple le plus connu est celui de la complexité cyclomatique de McCabe.

Il est aussi possible de mesurer la structure du code. Une programmation structurée est basée sur une décomposition du problème en sous-problèmes gérés par des composants du logiciel. Les mesures de couplages aident à déterminer si les composants du logiciel sont indépendants ou pas. On peut ainsi savoir grâce à cette indépendance si une modification d'un composant aura de lourdes répercutions sur le reste des autres composants.

Une dernière mesure du produit est celle de la documentation interne. Elle fournit aussi des informations. En effet, la documentation aide le programmeur à comprendre l'utilisation de telle ou telle variable ainsi que le but d'une méthode.

2.8.4 La mesure du service

On distingue deux catégories de mesures du service :

- l'entente de service ;

- l’étalonnage de la maintenance du logiciel.

L'entente de service

Définissons tout d’abord ce qu’est l’entente de service. L’entente de service, le ‘Service Level Agreement’ (SLA), est l’objectif de performance d’une organisation. Le SLA est donc un élément important de la satisfaction du client.

Pour y parvenir, les organisations séparent souvent les activités de support en rôles bien définis, permettant ainsi la spécialisation d’une équipe dans chacune des activités. On retrouve ainsi une partie du personnel s’occupant du client, une autre partie s’occupant de la correction et une autre du perfectionnement du logiciel.

Dans la plupart des situations, on trouve 3 niveaux [12] :

- Level 1 : l’équipe de « help desk » n’a pas de compétences techniques, et a seulement la responsabilité de recevoir les problèmes et de savoir quel technicien de support pourra aider l’utilisateur.
- Level 2 : les techniciens de support savent comment communiquer avec l’utilisateur, comprendre leur problème et les conseiller.
- Level 3 : les mainteneurs sont autorisés à modifier le produit.

L’étalonnage de la maintenance

L’étalonnage, plus connu sous sa traduction anglophone ‘benchmarking’, consiste pour une entreprise en une comparaison. Il faut donc, pour une entreprise, avant toute comparaison, pouvoir se connaître (ses propres processus, etc.). Il est possible de faire une comparaison interne (entre les secteurs de l’organisation), une comparaison externe avec les compétiteurs en ayant recours à une firme spécialisée dans l’étalonnage.

Voici quelques exemples de mesures comparées [11] :

- Points de fonctions supportés par personne (développement maison).
- Points de fonctions supportés par personne (développement maison + progiciels).
- Coût par point de fonction supporté.
- Nombre de langages de programmation supporté.
- % de type de programmation (Maintenance, Améliorations, Nouvelles applications).

2.9 Conclusion

Nous voyons que la maintenance est à présent une discipline à part entière avec son propre cycle de vie, indépendant de celui du développement ; elle a un processus propre avec des étapes bien définies, des activités qui lui sont spécifiques comme la gestion d’une file d’attente des requêtes. Elle peut être effectuée par les développeurs ou bien par une organisation indépendante.

La maintenance a également une ontologie propre, avec des facteurs l’influençant comme le personnel, le produit, etc.

Le problème majeur en maintenance de logiciel n’est pas de nature technique, c’est bien sa gestion qui est mise en cause. Il faut gérer une file d’attente de requêtes arrivant aléatoirement, avec peu d’outils spécifiques à la maintenance ; elle est perçue comme moins

importante et la tâche des développeurs est beaucoup plus difficile lorsqu'ils n'ont pas effectué un travail de qualité. Enfin la maintenance est aussi vue, à tort, comme coûteuse et lente. Mais c'est sans compter sur un comportement parfois ingérable des utilisateurs et sur une méconnaissance de la maintenance et de ce qui relève de son champ d'action.

La mesure en maintenance est aussi très importante, elle permet de garder le contrôle de la maintenance de logiciel grâce dans un premier temps à l'estimation des ressources. Dans ce cas, l'expérience peut être utile, mais il existe aussi des modèles. L'estimation des ressources n'est pas le seul outil de mesure. La mesure du processus de la maintenance ainsi que son produit nous fournissent des informations sur la 'maintenabilité' d'un logiciel, ou de l'effort à fournir pour le maintenir. Enfin, la mesure du service est aussi importante, d'une part en se mesurant par rapport aux autres, d'autre part en portant les yeux sur soi-même par une mesure de la capacité à satisfaire l'utilisateur.

Bien que la maintenance ait beaucoup de liens avec le développement de logiciel, son processus est différent, ses résultats le sont également. Elle peut reprendre beaucoup d'outils utiles au processus de développement, mais l'utilisation est contextuellement différente. *SM*^{Xpert} se veut un outil propre à la maintenance et est la preuve matérielle de l'existence de cette activité en génie logiciel.

Partie II

Chapitre 3 : SMXpert.

3.1 Introduction

Le nom de ce logiciel trouve ses racines dans son propre domaine d'application qu'est la maintenance. SM signifie Software Maintenance, le reste du mot signifie qu'il s'agit d'un système expert.

Ainsi qu'on l'a évoqué précédemment, l'objectif premier est d'améliorer l'interface, et en particulier l'interface des utilisateurs de type « expert » du logiciel *SM*^{Xpert}. Alors, avant toute analyse de problèmes, toutes critiques ainsi que tous changements, il faut présenter le logiciel SMXpert, d'où il vient et quel est son but.

3.2 Origine

SMXpert est l'aboutissement d'un travail de restructuration du logiciel COSMICXpert effectué par S. Sandron et B. Vanderose, après un travail préalable de familiarisation par le biais d'une phase de tests et de correction.

Le nom COSMICXpert a pour origine la méthode de mesure fonctionnelle COSMIC-FFP (Common Software Measurement International Consortium- Full Functional Point), les lettres C,O,S,M,I et C ont la même signification et le reste du mot signifie, comme pour *SM*^{Xpert}, que c'est un système expert.

Il y a deux grandes distinctions entre COSMICXpert et *SM*^{Xpert}. D'une part, le domaine de connaissances de COSMICXpert a été changé pour s'appliquer désormais à l'évaluation de la maturité des processus de maintenance de logiciel, d'autre part, l'architecture de

COSMICXpert a aussi été modifiée pour devenir celle de SM^{Xpert}. « Celle-ci a été repensée et adaptée aux nouveaux besoins. » [13]

L'objectif principal de COSMICXpert, logiciel à base de connaissance utilisant le raisonnement basé sur des cas, est d'offrir un apprentissage pour la méthode de mesure fonctionnelle *COSMIC-FFP*.

« De ce fait, la base de connaissances du logiciel repose sur les liens tissés entre une ontologie du génie logiciel et une ontologie de la mesure logicielle *COSMIC-FFP* .» [13]

L'ontologie choisie pour le génie logiciel est issue du SWEBOK (Software Engineering Body of Knowledge).

La méthode de mesure fonctionnelle *COSMIC-FFP* permet de mesurer la taille fonctionnelle d'un logiciel, en s'appuyant sur la méthode des Points de Fonctions.

COMSIC-FFP est le fruit des laboratoires de recherche du génie logiciel et de métrique de l'Université du Québec à Montréal (UQAM) ; elle « respecte les principes du Common Software Measurement International Consortium et est, depuis 2002 une norme ISO (19761). » [13] Il existe beaucoup de méthodes de mesure basées sur les points de fonctions. *COSMIC-FFP* a pour but de pallier aux nombreux manques de ces dernières et privilégie plus particulièrement le monde de l'industrie. *COSMIC-FFP* a pour domaine d'application [13] :

- Le domaine des logiciels d'affaires, tels que les applications bancaires, de production ou de facturation, etc.
- Le domaine des logiciels en temps réel conçus pour garder le contrôle des évènements du monde réel, tels que les applications liées aux télécoms, à l'aviation ou encore les systèmes d'exploitation.
- Les logiciels tenant des deux catégories précédentes, tels que les logiciels de réservation en temps réel d'avions et d'hôtels.

Le logiciel SM^{Xpert} est un logiciel à base de connaissance permettant de supporter l'usage du Modèle d'Evaluation de la Capacité à Maintenir le Logiciel (S^{3m}). Modèle conçu pour améliorer la maintenance, ainsi que pallier aux insuffisances des modèles existants.

Lors de leur démarche de restructuration S. Sandron et B. Vandenrose ont dû constater que le domaine d'application de COSMICXpert et SM^{Xpert}, qui est la maintenance, sont très différents, ce qui rendait à première vue la démarche très délicate.

En effet, S. Sandron et B. Vandenrose [13] ont souligné que

- Les deux domaines de connaissances sont différents par la taille. L'ontologie de la maintenance est beaucoup plus vaste dans ses concepts que celle de COSMIC-FFP.
- Les deux domaines de connaissances sont différents par le contenu. La mesure se concentre sur les produits des différentes phases du cycle de vie de développement de logiciel. Tandis que la maintenance ne se focalise pas uniquement sur le produit du développement, mais aussi sur la gestion des demandes, etc.

Ainsi au stade de la comparaison des domaines, et avant leur démarche de restructuration, rien n'indiquait que l'adaptation du logiciel COSMICXpert au nouveau domaine qu'est la maintenance, était possible. S. Sandron et B. Vandenrose ont alors élaboré le schéma conceptuel du logiciel SM^{Xpert} à partir de l'ontologie de la maintenance et du Modèle d'Evaluation de la Capacité à Maintenir le Logiciel. Et c'est à l'issue de l'observation des

schémas conceptuels respectifs qu'ils ont décidé que leur démarche de restructuration était possible, car ces derniers étaient similaires.

3.3 Buts

Le logiciel *SM*^{Xpert} est un système expert permettant d'aider un utilisateur à évaluer la maturité d'un processus de maintenance et de résoudre les problèmes de celui-ci. « De part l'étendue de ce domaine de connaissances, *SM*^{Xpert} s'adresse à un large panel d'utilisateurs. » [13] Les utilisateurs peuvent très bien être des évaluateurs, soucieux d'apprécier la qualité de leur processus de maintenance, ou bien des ingénieurs de la maintenance désirant résoudre un problème dans leur processus de maintenance. Mais la portée du logiciel ne s'arrête pas là. Comme nous l'avons vu, le processus de maintenance commence avant la livraison d'un logiciel et les développeurs peuvent aussi être des mainteneurs. Le logiciel *SM*^{Xpert} peut donc aussi leur être utile.

3.4 Le logiciel

A présent que nous savons d'où vient le logiciel *SM*^{Xpert}, que nous avons déjà expliqué son domaine d'application qu'est la maintenance et que nous savons son but, il est intéressant d'évoquer le logiciel proprement dit.

Pour présenter le logiciel *SM*^{Xpert}, quoi de plus naturel que de donner sa spécification. Toutefois une spécification exhaustive n'est pas nécessaire, seules ici les informations pertinentes seront présentées. Pour plus d'information, le lecteur est amené à lire [13] chapitre 2, 3 et 4.

Commençons par le profil des utilisateurs. Le logiciel *SM*^{Xpert} présente trois modes d'interactions pour trois types d'utilisateurs différents : l'utilisateur de type « user », l'utilisateur de type « expert » et l'administrateur. Voici plus en détail chacun de ces types d'utilisateurs [13] :

- **L'administrateur :** peut gérer le profil de tous les utilisateurs.
- **L'utilisateur de type « expert » :** peut faire évoluer la base des connaissances du logiciel. Par exemple : insérer de nouveau cas problèmes. Il peut également calibrer les connaissances afin qu'elles soient présentées uniquement à un utilisateur de type « user » dont le profil (en réalité celui de son entreprise) est adapté.
- **L'utilisateur de type « user » :** peut consulter la base de connaissances et être assisté dans la tâche d'évaluation de la maturité d'un processus. L'utilisateur de type « user » dispose de l'accès à une seule interface : celle propre à son type d'utilisateur. Cette interface permet de mener à bien les tâches successives du processus suivi par un évaluateur afin de bien interpréter les pratiques de (*S*^{3m}).

A présent, présentons le schéma conceptuel du logiciel *SM*^{Xpert}, ce schéma résulte de l'ontologie de la maintenance du logiciel et l'ontologie du Modèle d'Evaluation de la Capacité à Maintenir le Logiciel.

Voici le schéma conceptuel ainsi qu'une explication des concepts [13] :

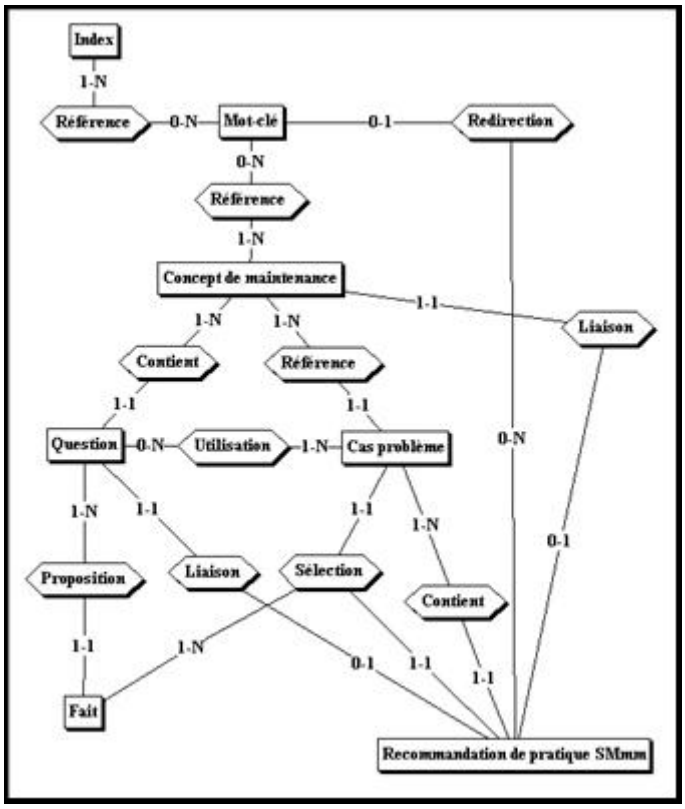


Figure 18 : Schéma conceptuel de SMXpert [13]

Les concepts de maintenance sont des concepts inhérents à la problématique de la maintenance du logiciel. Ils établissent un lien entre les problèmes rencontrés par les utilisateurs ou développeurs et les connaissances fondamentales de la maintenance du logiciel.

Un mot-clé est n'importe quel mot du domaine de la maintenance du logiciel pouvant être associé à un concept de maintenance.

Un mot d'index est un terme du génie logiciel plus générique et donc moins précis qu'un mot-clé mais permettant d'orienter l'utilisateur vers le mot-clé adapté à sa question.

Un cas problème est un cas pour lequel l'évaluateur doit utiliser un processus de diagnostic afin d'identifier correctement une pratique de l'ontologie de (S^{3m}).

Les questions (thèmes) sont les pistes de réflexion à suivre par l'évaluateur pour trouver une solution aux cas problèmes.

Un fait est la réponse de l'évaluateur en rapport à une question et au cas problème relié à cette question.

Une recommandation est la proposition de solution fournie par le système ou des indications permettant à l'évaluateur de raffiner sa solution (par exemple la redirection vers un autre mot-clé).

Afin d'illustrer plus encore, voici un schéma du déroulement :

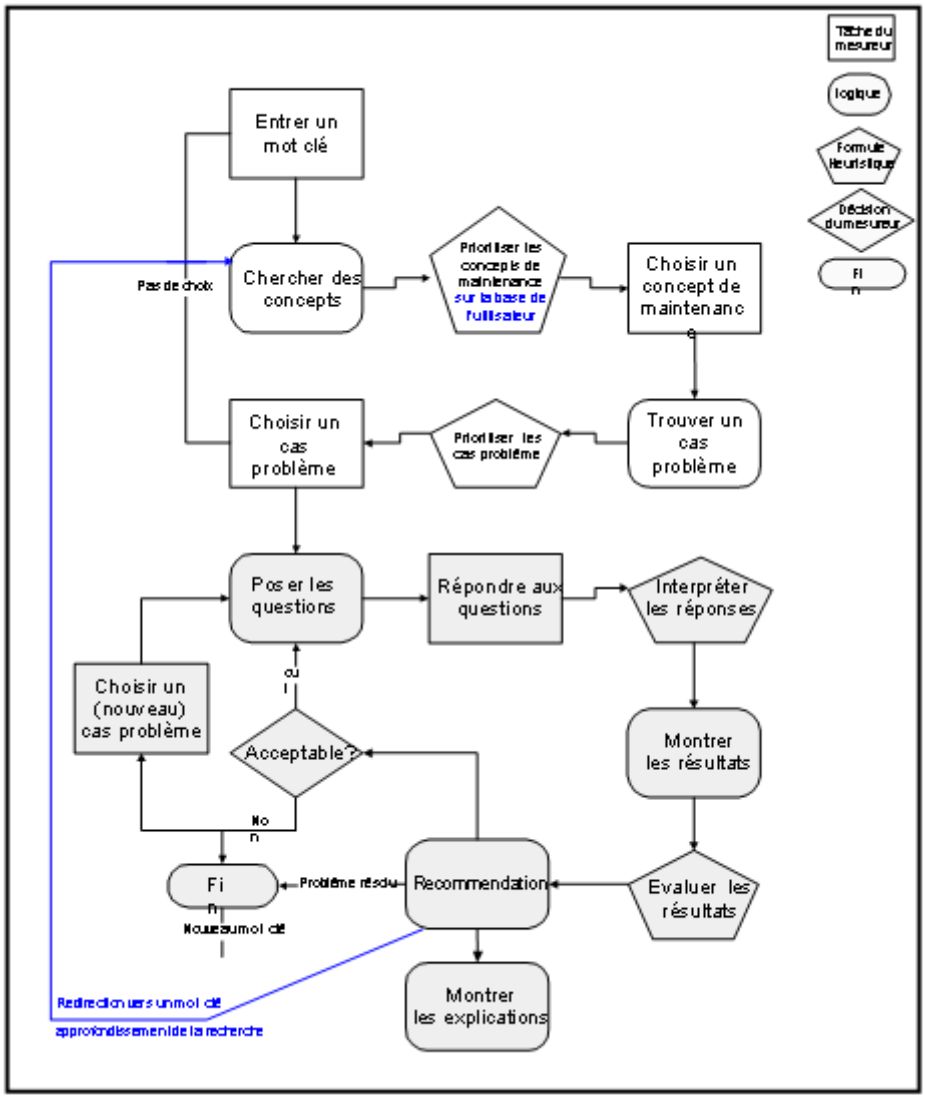


Figure 19 : Enchaînement des tâches de l'évaluation de la maturité dans *SM*^{Xpert} [13]

Grâce au schéma du déroulement, nous pouvons expliquer le type de raisonnement du logiciel. Le logiciel *SM*^{Xpert} pose deux types de raisonnement : le raisonnement pas cas et le raisonnement par règles. « Les premières étapes du raisonnement, jusqu'à l'affichage des questions, relèvent du raisonnement par cas. » « Tandis que le choix de la bonne recommandation sur base des réponses aux questions dénote un raisonnement par application de règles. » [13] Les questions posées par le système sont posées en même temps et leur ordre a une importance, car le système fournira la recommandation suivant la première question à laquelle l'utilisateur répondra « oui ». Le moteur d'inférence est donc un mécanisme de recherche d'une recommandation suivant une réponse donnée à une question.

Un autre fait important du déroulement est le mécanisme de redirection. Les recommandations peuvent amener l'utilisateur à approfondir sa rechercher en étant redirigé. Ce mécanisme est en fait l'entrée d'un mot-clé, par le système, à la place de l'utilisateur. Ce mot-clé étant évidemment choisi par un expert en maintenance et mémorisé lors d'une opération d'évolution de la base des connaissances.

Il faut encore définir l'architecture logique abstraite et concrète, ainsi que l'architecture physique du logiciel *SM*^{Xpert}, ces définitions se feront sur base d'illustrations.

La Figure 20 illustre l'architecture logique abstraite de *SM*^{Xpert}. On remarque que chaque type d'utilisateur dispose d'une interface qui lui est propre. « Ces interfaces ne sont que des vues permettant d'interagir avec le système. La seule complexité à ce niveau provient du fait que chaque interface est responsable de la correction syntaxique des données qu'elle fournit au système, d'où l'existence d'un module *validation syntaxique.* » [13]

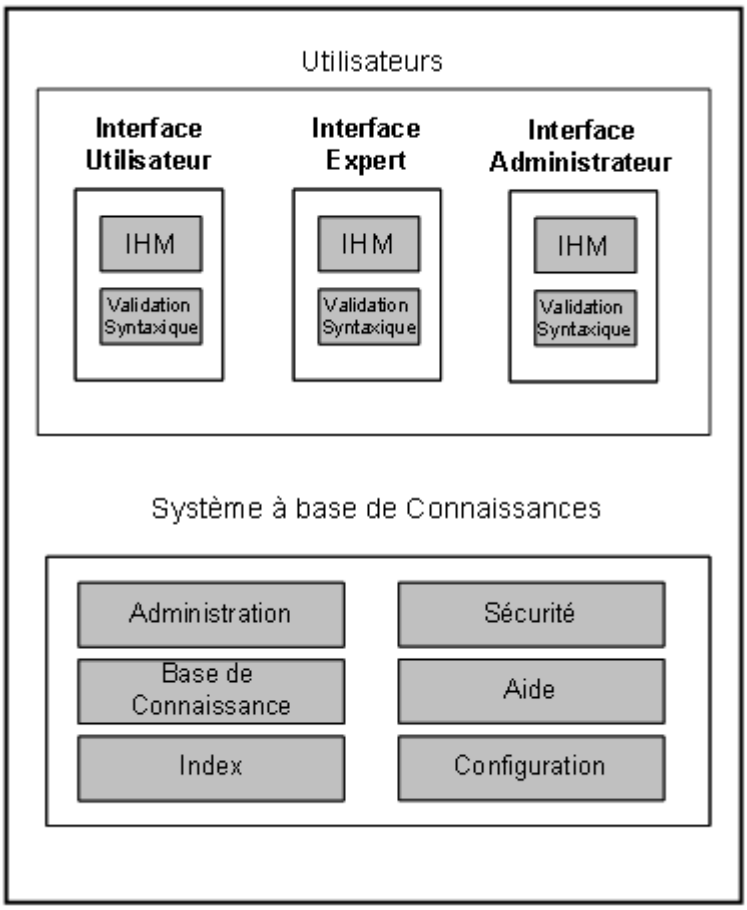


Figure 20 : architecture logique abstraite de *SM*^{Xpert} [13]

Le système à base de connaissance comporte six éléments voici une illustration (Figure 21) plus détaillée de ces six composants permettant de comprendre leur fonctionnalité.

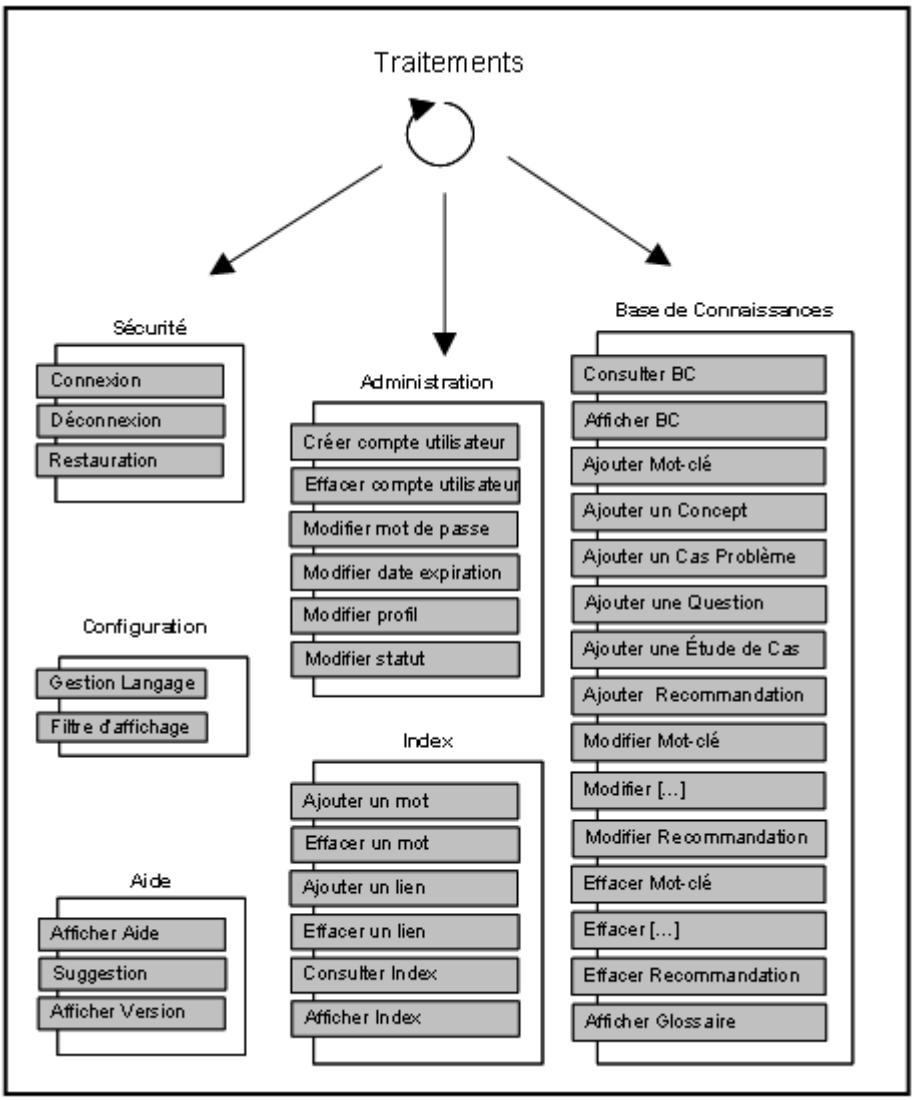


Figure 21 : Schéma des composants [13]

Exposons à présent l'architecture logique concrète. Le logiciel *SM*^{Xpert} est une application Web, c'est-à-dire un programme situé sur un serveur Web. *SM*^{Xpert} utilise des technologies lui permettant d'être accessible à partir d'Internet. Une explication des technologies servant à la création de *SM*^{Xpert} est donnée dans [13] chapitre 4 :

Le *design pattern* convenant le mieux pour une application Web est le « Model – View – Controller ». L'architecture logique concrète suit ce *design pattern* (Figure 22).

Les information sur le *design pattern* « Model – View – Controller » sont expliquées à la section 4.4.1 de [13] :

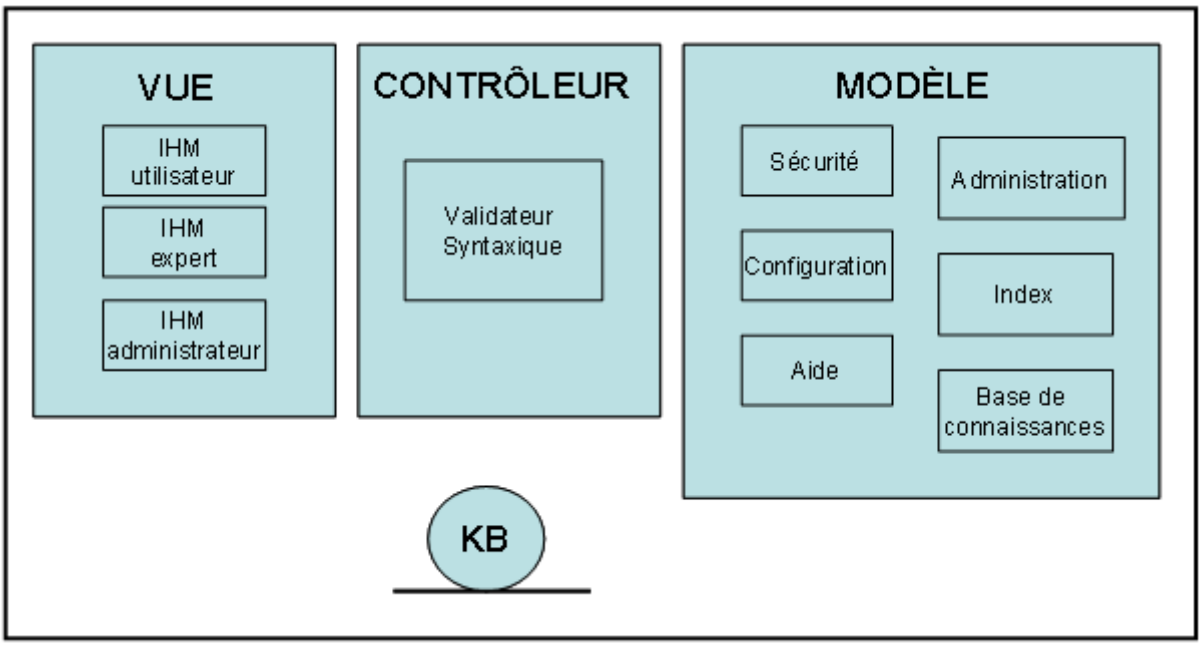


Figure 22 : Architecture logique concrète de *SM*^{Xpert} [13]

Sur ce schéma, la place du validateur syntaxique se trouve dans le contrôleur, afin de ne pas faire transiter des données erronées entre la vue et le modèle. L'élément « KB » correspond à la base de connaissance.

Présentons enfin l'architecture physique du logiciel *SM*^{Xpert}, c'est une architecture « multi-tiers », il y a une séparation physique des composants.

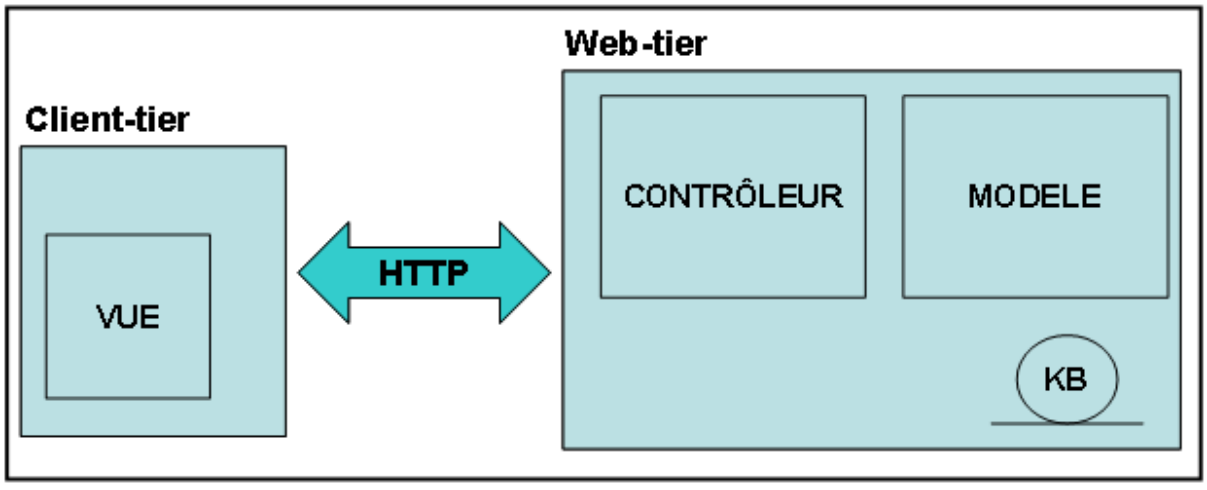


Figure 23 : Architecture physique de *SM*^{Xpert} [13]

- Le « Client-tier » correspond au navigateur utilisé par un utilisateur et à l'ensemble des interfaces. Tout cela constitue la vue.
- Le « Server-tier », c'est l'application en tant que telle ; il traite les requêtes en provenance de l'utilisateur et envoie les réponses qui seront affichées sur les interfaces de l'utilisateur. C'est également lui qui gère les accès à la base de connaissances.

3.5 Conclusion

SM^{Xpert} est un logiciel d'aide à la décision pour évaluer la maturité d'un processus de maintenance et résoudre les problèmes de celui-ci. Il est le résultat d'un travail de restructuration, effectué par messieurs S. Sandron et B. Vanderose, du logiciel COSMICXpert. La maintenance de logiciel, qui allait être le domaine d'application du logiciel *SM*^{Xpert}, présentait de grosses différences avec le domaine d'application du logiciel COSMICXpert de par sa taille et son contenu. Mais les schéma conceptuels étaient très proches, ce qui a permis le travail de restructuration et donné naissance au logiciel *SM*^{Xpert}, un logiciel hybride combinant le raisonnement par cas et le raisonnement par règle.

Chapitre 4 Faiblesses de *SM*^{Xpert}

4.1 Introduction

Cette section présente la mise à jour des faiblesses de *SM*^{Xpert}, à partir des exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation. Nous allons voir les problèmes d'interface mais aussi des problèmes de nature conceptuels.

4.2 Analyse

Analysons comment le logiciel SMXpert se conforme au niveau des critères portant sur les principes d'ergonomie et des critères portant sur la présentation de l'information. Les deux interfaces principales du logiciel SMXpert, l'interface des utilisateurs de type « user » et de type « expert », seront confrontées à chacun des critères. Lorsqu'une des interfaces ne sera pas en accord avec un critère, un exemple (ou plusieurs), en rapport avec les fonctionnalités principales du logiciel, sera ajouté pour appuyer cette constatation. Il se peut que certains critères ne puissent pas être confrontés à la version présentement critiquée du logiciel SMXpert ou à une des deux interfaces principales

4.2.1 Principes de dialogue

Adaptation de la tâche

1. Présenter l'information liée à l'exécution d'une tâche.

La page pour les utilisateurs de type « user » est en accord avec ce critère.

Cette page servant à naviguer dans la connaissance, l'information présentée concerne cette fonctionnalité, comme l'illustre la figure (la première de l'illustration de l'interface des utilisateurs de type « user »).

Tandis que l'interface des utilisateurs de type « expert » n'est pas conforme à ce critère, et ce dès la page principale de l'interface. En effet, les concepts de maintenance ainsi que les mots-clés et cas problèmes associés sont sur cette page, tout ceci fait partie du contenu de la base de connaissance. Il y a aussi un menu au-dessus, avec un onglet (Xpert trees) permettant d'avoir accès au reste de la base de connaissance, mais les autres onglets du menu permettent d'avoir accès à d'autres pages (ex : la page pour les utilisateur de type « user ») et d'autres fonctions.

Cela veut dire qu'il y a des boutons de navigation dans le logiciel, de navigation dans la base des connaissances et une partie de la base de connaissance sur une même page. Cela n'est pas conforme avec le critère.

La Figure 24 suivante illustre cela. La capture d'écran de la page d'accueil de l'interface des utilisateurs de type « expert » est impossible à réaliser, tant elle est grande de par le contenu de son information. On peut avoir une brève idée en voyant que huit concepts de maintenance (« Maintenance Concept ») et leurs mots-clés (« Keywords ») respectifs sont affichés. Bien évidement le nombre de mots-clés associés à un concept de maintenance varie, mais on voit que sur la Figure 24, le premier concept (« Changing Priorities ») il y a déjà trente-sept mots-clés associés.

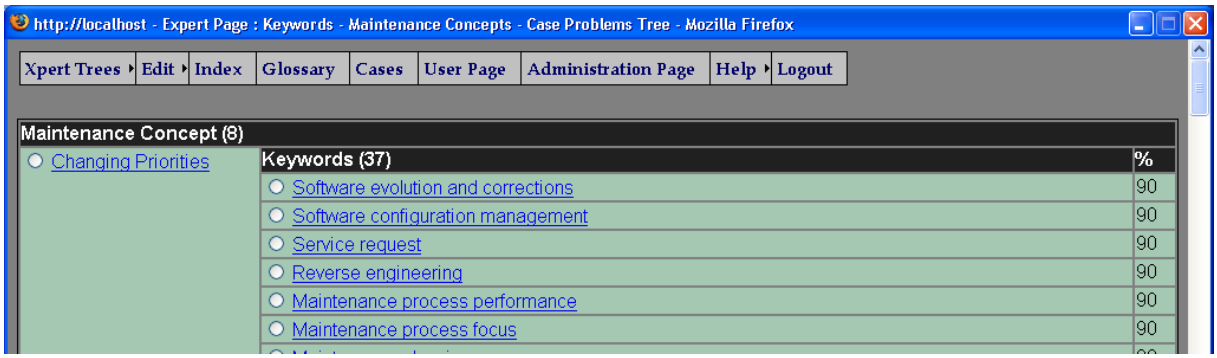


Figure 24 : Page d'accueil des utilisateurs de type « expert »

Attention que le type de navigation dans la base des connaissance de la page des utilisateurs de type « expert » n'est pas similaire au type de navigation de la page des utilisateurs de type « user » ; dans ce cas-ci c'est une navigation permettant de modifier la base des connaissances.

2. L'aide doit toujours être disponible et liée à la tâche en cours.

L'aide est disponible aussi bien pour l'interface des utilisateurs de type « expert » que pour l'interface des utilisateurs de types « user », mais n'est pas nécessairement liée à la tâche en cours. En effet, l'aide de la page des utilisateurs de types « user » fournit de l'information sur ce qu'est un mot-clé, et non sur la manière d'utiliser la page.

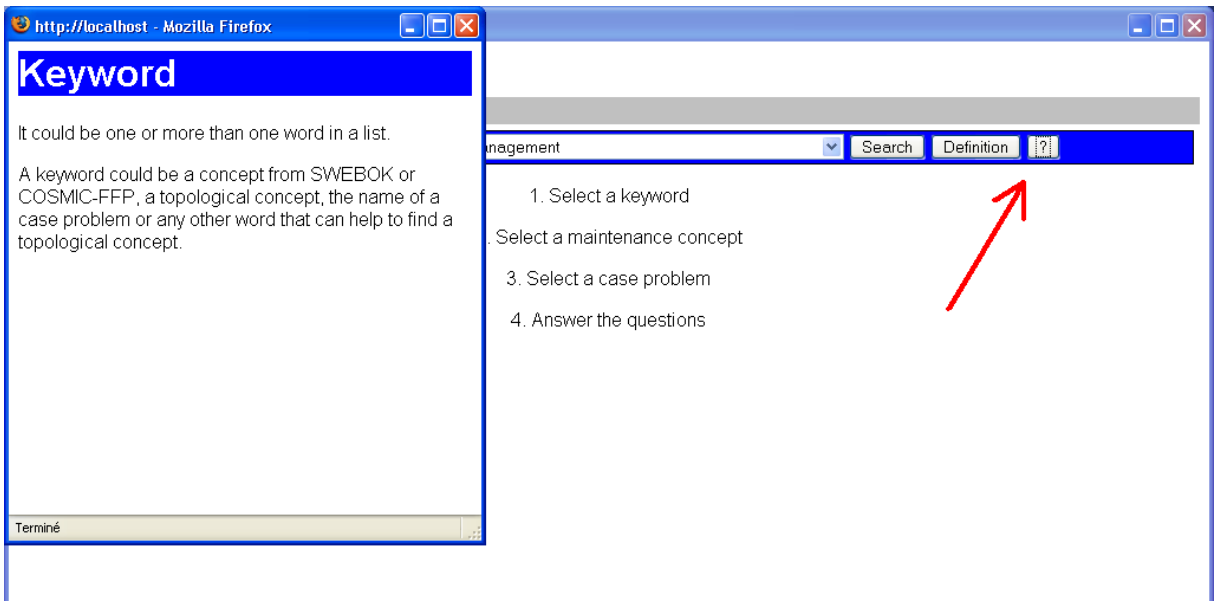


Figure 25 : Aide de la page des utilisateurs de type « user »

Pour les pages des utilisateurs de types « expert », l'aide est accessible à partir de toutes les pages, mais est toujours la même et fournit de l'information sur les mots-clés, concepts

topologiques, etc. Cette information n'explique pas chaque élément, mais plutôt son utilité dans la page des utilisateurs de type « user ».

La Figure 26 l'illustre. Attention que le mot concept topologique n'existe pas dans le logiciel *SM*^{Xpert}. Il existe dans le logiciel COSMICXpert, c'est un oubli de modification lors de la phase de restructuration.

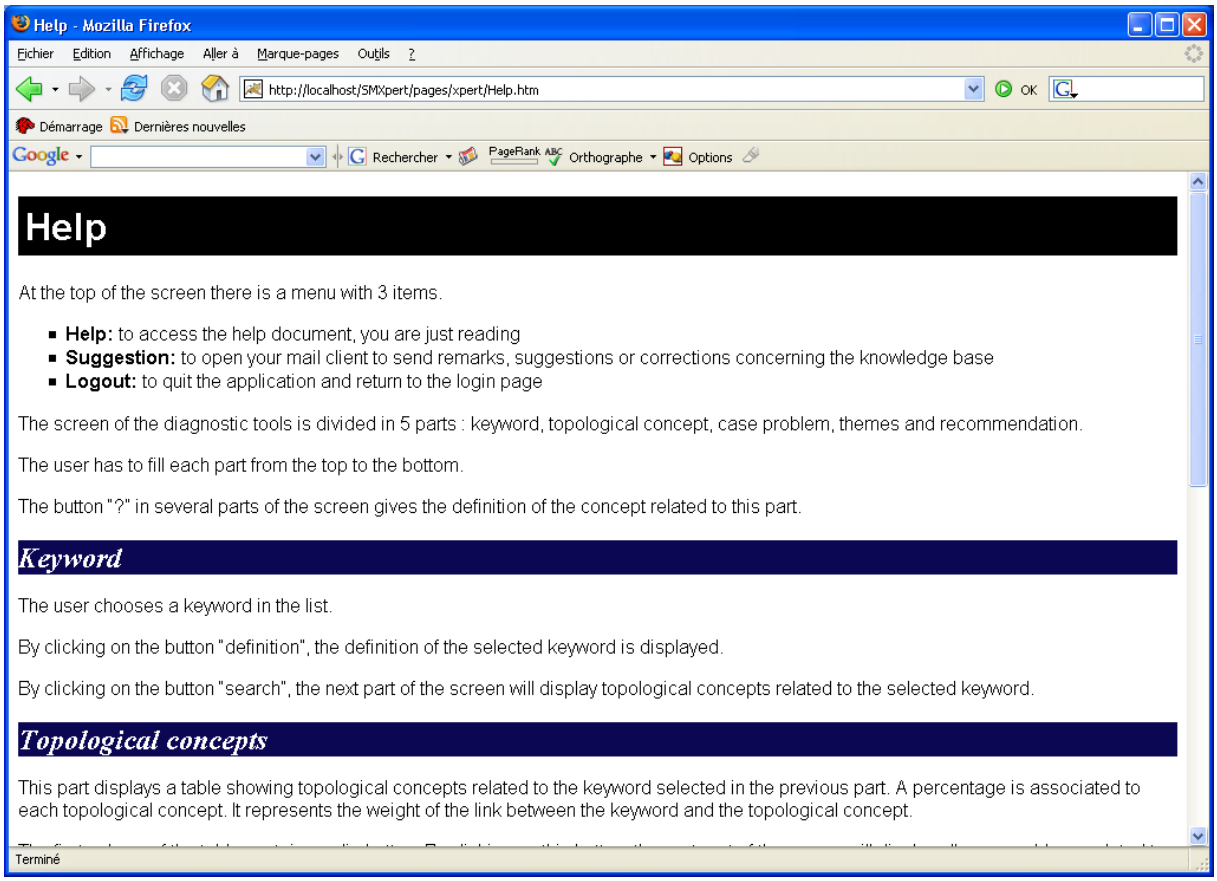


Figure 26 : L'aide de la page des utilisateurs de type « expert »

Aucune aide supplémentaire ne survient lors de l'utilisation du logiciel. Mise à part la page des utilisateurs de type « user », lorsqu'un sous-menu apparaît, une aide est fournie pour ce sous-menu, mais elle constitue plus une définition du titre du sous-menu qu'une explication de son utilité et de ses possibilités. *SM*^{Xpert} ne remplit donc pas ce critère.

La Figure 27 illustre ceci par l'exemple du choix d'un mot-clé (« Change Control ») et de l'affichage, dans un sous-menu, des concepts de maintenance qui lui sont liés. Le bouton d'aide à côté du mot « Maintenance Concept » a été cliqué, et normalement la définition doit apparaître, mise à par qu'ici également l'oubli, lors de la phase de restructuration a eu lieu.

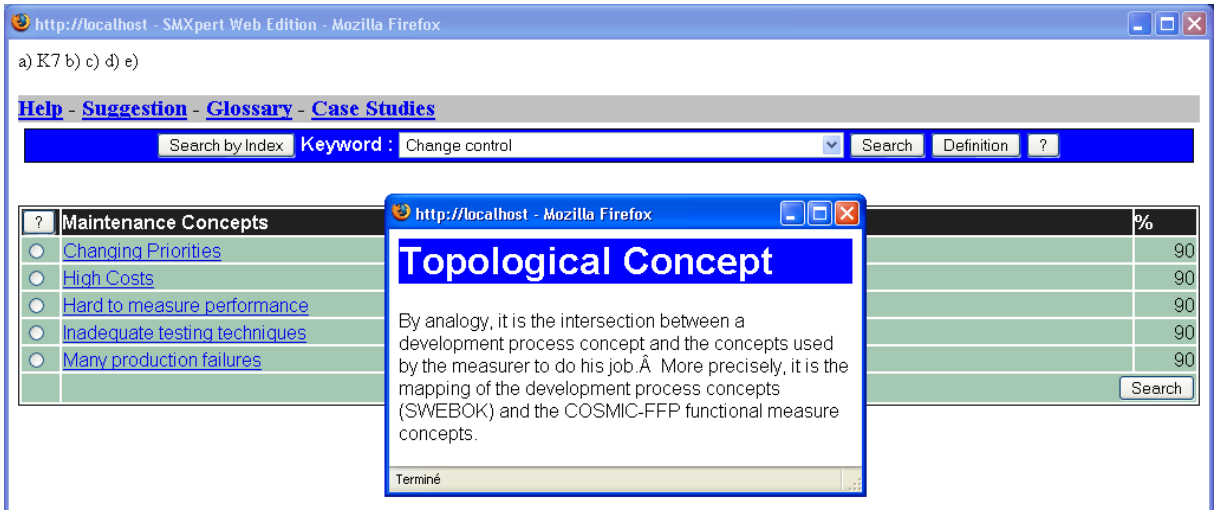


Figure 27 : Aide des sous-menus sur la page des utilisateurs de type « user »

3. Toutes actions pouvant être automatisée doivent être faites par le logiciel.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. Prenons par exemple la fonction de filtre qui permet de sélectionner l'information à afficher. Le filtre est réglable à partir du menu des pages des utilisateurs de type « expert ». Il est donc impossible pour un utilisateur de type « user » de régler les filtres pour que l'information lui soit fournie.

Mais il est surtout impossible de le sauvegarder, ce filtre n'est valable que pour une session sur le logiciel *SM*^{Xpert}. Il faut donc à chaque session régler les filtres.

4. Prise en compte de la complexité de la tâche (ex : utiliser un menu offrant les choix possibles lorsqu'il y a un ensemble d'entrées d'alternatives).

Ce critère est respecté par le logiciel *SM*^{Xpert}.

5. L'utilisateur ne doit pas entrer des valeurs pas défaut, s'il en existe.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

Le seul cas trouvé de valeur par défaut à ne pas entrer se situe dans la procédure d'entrée d'un cas, lorsqu'il est demandé de lier un cas avec un cas d'étude ; s'il n'existe qu'un cas d'étude alors celui-ci est déjà sélectionné.

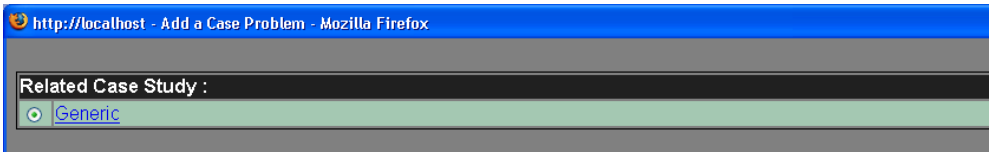


Figure 28 : Exemple de valeur par défaut

Caractère auto descriptif

1. Fournir un compte rendu, aux actions de l'utilisateur, qui se réfère strictement à la situation pour laquelle il est nécessaire.

Le logiciel *SM*^{Xpert} ne remplit pas du tout ce critère. Aucun compte rendu n'est fourni à l'utilisateur.

2. Si des conséquences graves peuvent résulter d'une des actions de l'utilisateur, une explication doit lui être fournie ainsi qu'une demande de confirmation.

Le logiciel *SM*^{Xpert} ne remplit pas ce critère.

Par exemple, lors d'un changement de définition de mot-clé, il n'y a aucune demande de confirmation. Lors de l'insertion d'un cas problème il ne nous est pas non plus demandé de confirmer l'insertion du cas problème.

En règle générale, il n'est jamais proposé de demande de confirmation lors d'un changement (modification, suppression, ajout) de la base des connaissances.

3. Le compte rendu et les explications doivent être adaptés au niveau de connaissance de l'utilisateur.

Le logiciel *SM*^{Xpert} ne remplit pas ce critère. Etant donné qu'aucun compte rendu ni explication ne sont fournis, c'est impossible.

4. Le compte rendu et les explications se doivent d'être plus ou moins détaillés selon les besoins de l'utilisateur.

Le logiciel *SM*^{Xpert} ne remplit pas ce critère. Etant donné qu'aucun compte rendu ni explication ne sont fournis, c'est impossible.

5. Lorsqu'il faut entrer des données, il convient que le logiciel indique à l'utilisateur la nature des données à entrer.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

Contrôle explicite

1. Si le dialogue a été interrompu, il convient que l'utilisateur ait la possibilité de déterminer à quel endroit le reprendre chaque fois que la tâche le permet.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}.

Par exemple, pour la procédure d'entrée d'un cas problème, si un incident intervient, il faut recommencer toute la procédure depuis le début.

S'il y a une erreur ou un manquement pour un des champs d'une étape de la procédure, l'utilisateur est averti de ce manquement après la procédure. Le logiciel lui permet de revenir à l'endroit où l'erreur s'est produite, mais le logiciel n'a pas sauvegardé ce qui suivait cette erreur. Ainsi l'utilisateur doit recommencer la procédure à partir de l'endroit où il a fait une erreur et non seulement rectifier son erreur.

2. L'utilisateur doit pouvoir annuler au moins la dernière étape d'un dialogue.

Ceci n'est pas respecté entièrement par le logiciel *SM*^{Xpert}.

Sur la page des utilisateurs de type « user » il n'est pas possible d'annuler une étape, comme le choix d'un mot-clé.

De même, pour l'interface des utilisateurs de type « expert », il est seulement possible à certaines étapes de la procédure d'entrée d'un cas problème d'annuler complètement la procédure mais pas la dernière étape.

3. Les méthodes d'interaction doivent être adaptées aux besoins et aux caractéristiques de l'utilisateur.

Le logiciel *SM*^{Xpert} peut être utilisé par deux types d'utilisateurs : les utilisateurs de type « expert » et les utilisateurs de type « user ». On ne prendra pas en compte ici les utilisateurs de type « administrateur ».

On pourrait croire qu'avec cette division le logiciel *SM*^{Xpert} remplit cette fonction.

Mais il n'est pas possible pour un utilisateur commun de s'enregistrer sur le système, il lui faut pour cela avoir recours à un administrateur du logiciel.

Il existe plusieurs niveaux d'utilisateurs, et l'information à leur fournir doit être différente suivant le niveau de l'utilisateur. Mais le logiciel ne prend pas encore en compte les différents niveaux d'utilisateurs.

Facilité d'apprentissage

1. Il convient que l'utilisateur dispose des règles et concepts sous-jacents utiles à l'apprentissage.

Ce critère est respecté par le logiciel *SM*^{Xpert} dans la mesure où il est possible de trouver des règles utiles à l'apprentissage. Toutefois, ces règles se trouvent dans le menu d'aide de la page des utilisateurs de type « expert » et concernent la navigation dans la base des connaissances pour la page des utilisateurs de type « user ». Il n'existe par contre aucune règle ni concepts sous-jacents utiles à l'apprentissage pour l'interface des utilisateurs de type « expert ».

Dans la procédure d'entrée d'un cas, beaucoup d'informations sont demandées à l'utilisateur, mais jamais il n'est expliqué pourquoi ces informations sont demandées, ni leur(s) utilité(s).

De plus, même si cela était fait, la procédure d'entrée d'un cas ne serait pas suffisante pour comprendre les concepts sous-jacents, car la procédure est incomplète ; une fois faite, il reste quelques manipulations à réaliser. Après l'entrée des recommandations, il faut encore relier les questions avec les recommandations. On pourrait dire que la procédure est divisée en deux grosses parties, mais ceci doit néanmoins être deviné par l'expert.

Tolérance aux erreurs

1. L'application doit assister l'utilisateur dans la prévention et la détection d'erreurs d'entrée.

Le logiciel *SM*^{Xpert} remplit en partie ce critère.

Lors de la procédure d'entrée d'un cas, plusieurs étapes sont à réaliser ; si une erreur ou un manquement survient lors d'une étape, l'utilisateur est averti à la fin de la procédure.

Le logiciel lui permet de revenir à l'endroit posant problème.

2. Le système doit aussi empêcher qu'une erreur d'entrée provoque des résultats non prédéfinis et des défaillances du système.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. En effet, l'insertion de certains caractères dans les champs de saisie, comme le « é », n'est pas reconnue par le système. Le logiciel n'avertit pas l'utilisateur que ce caractère n'est pas reconnu par le système.

Par exemple, lors d'une procédure d'entrée d'un cas, si un caractère n'est pas reconnu par le logiciel, en fonction du caractère et de l'endroit où il a été inséré, le logiciel s'arrêtera… ou alors continuera, mais il sera impossible de consulter, par le biais de la navigation dans la base des connaissances, ce que a été inséré. Soit le logiciel s'arrêtera, soit il n'affichera rien.

3. Dans le cas où le système peut corriger des erreurs automatiquement, il convient que le système avertisse l'utilisateur de l'exécution des corrections et qu'il laisse la possibilité d'ignorer les corrections.

Le logiciel *SM*^{Xpert} ne permet pas à l'utilisateur de faire beaucoup d'erreurs, mise à part l'entrée de caractères non reconnus ou encore l'entrée de données non numériques dans les champs qui l'exigent. Il est évidemment impossible pour le logiciel de corriger automatiquement ce genre d'erreur, il ne peut que la signaler.

Flexibilité

1. Il convient que le logiciel permette à l'utilisateur de choisir entre différentes formes de représentation, selon ses préférences et la complexité de l'information à traiter.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}.

La page des utilisateurs de types « user » est la même pour tous les utilisateurs et il n'est pas possible qu'un utilisateur puisse demander une autre disposition des sous-menus.

Il n'est pas possible de faire une navigation en demandant directement un concept de maintenance, il faut toujours faire la même démarche.

La Figure 29 montre toute la page d'accueil des utilisateurs de type « user ». Il n'est pas possible de l'adapter selon ses préférences.

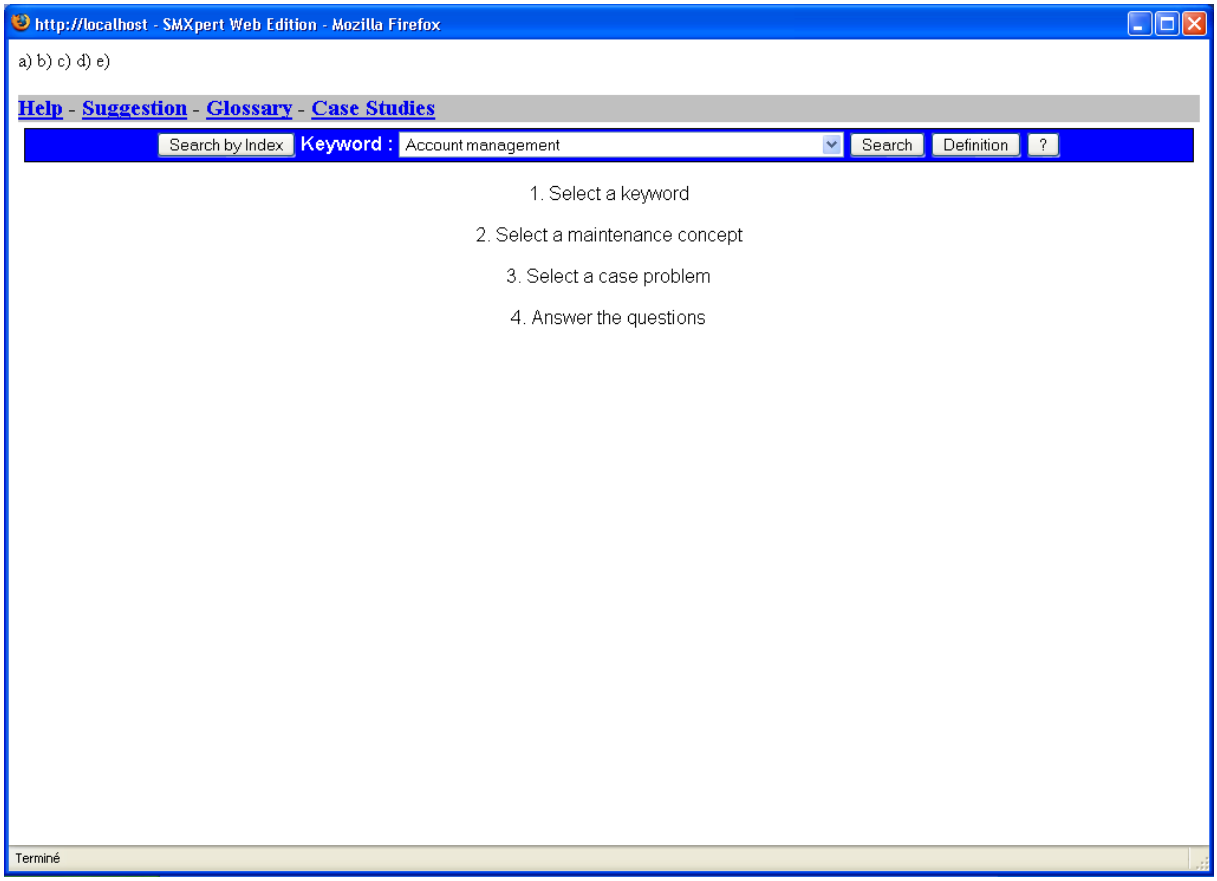


Figure 29 : Page d'accueil des utilisateurs de type « user »

Conformité aux attentes de l'utilisateur

1. L'application doit employer le vocabulaire familier de l'utilisateur.

Ce critère est rempli par le logiciel *SM*^{Xpert}.

2. Pour des tâches similaires, les dialogues utilisés doivent être similaires, pour que l'utilisateur puisse mettre au point des méthodes communes de résolution de problèmes relatifs à la tâche.

Il existe des tâches ayant le même but. Exemple : modifier, supprimer, ajouter des concepts de maintenances, mots-clés, cas problème etc.

Pour l'ajout d'un cas problème le dialogue est unique car il n'y a aucune autre tâche similaire. Mais, en ce qui concerne les mots-clés, les concept de maintenance, les mots d'index, les principes de dialogues sont similaires pour chacune de ces tâches, et les manipulations d'interface pour démarrer ces tâches sont toujours les mêmes. Il faut pour cela, dans la page des utilisateurs de type « expert » cliquer sur l'onglet « edit » et les diverses fonctionnalités deviennent accessibles.

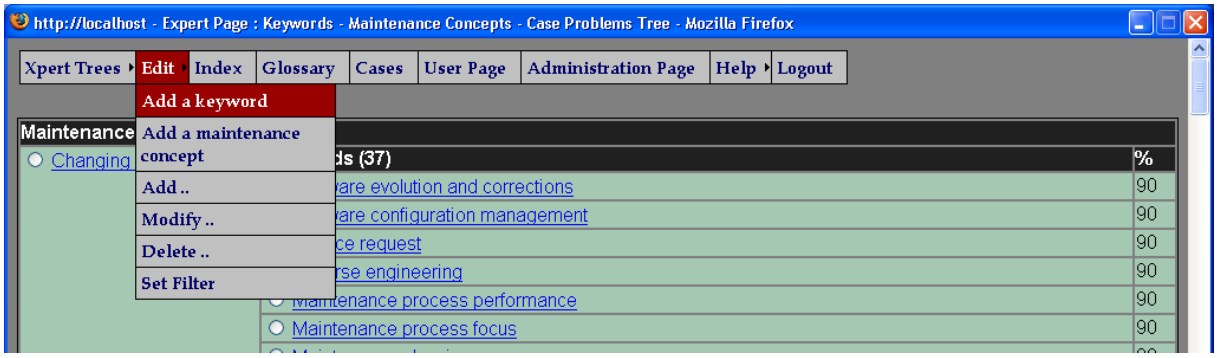


Figure 30 : Illustration de tâches similaires

On peut dire que le logiciel *SM*^{Xpert} remplit bien cette fonction.

4.2.2 Présentation de l'information 1/2

Les sections 4.2.2 et 4.2.3 sont les critères reprenant les recommandations concernant la présentation de l'information.

Cette section présente une évaluation des caractéristiques de l'information présentée et de l'adéquation des fenêtres dans l'organisation de l'information. La section suivante confronte le logiciel *SM*^{Xpert} aux recommandations relatives aux fenêtres, zones, groupes, etc.

1. Clarté

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}.

Par exemple, dans la page des utilisateurs de type « user », il n'est pas aisé d'avoir rapidement l'information désirée, qui est une recommandation.

En effet, il faut d'abord choisir un mot-clé, puis choisir un concept de maintenance ; il faut choisir ensuite un cas problème et finalement répondre à des thèmes. Suivant les réponses aux différents thèmes, le logiciel fournit un lien (Figure 31) pour accéder à la page où se trouve la recommandation ainsi qu'un bouton pour être redirigé vers une nouvelle navigation si la recommandation obtenue (et non encore lue) préconise de faire une navigation supplémentaire avec un autre mot-clé afin d'obtenir de plus amples informations.

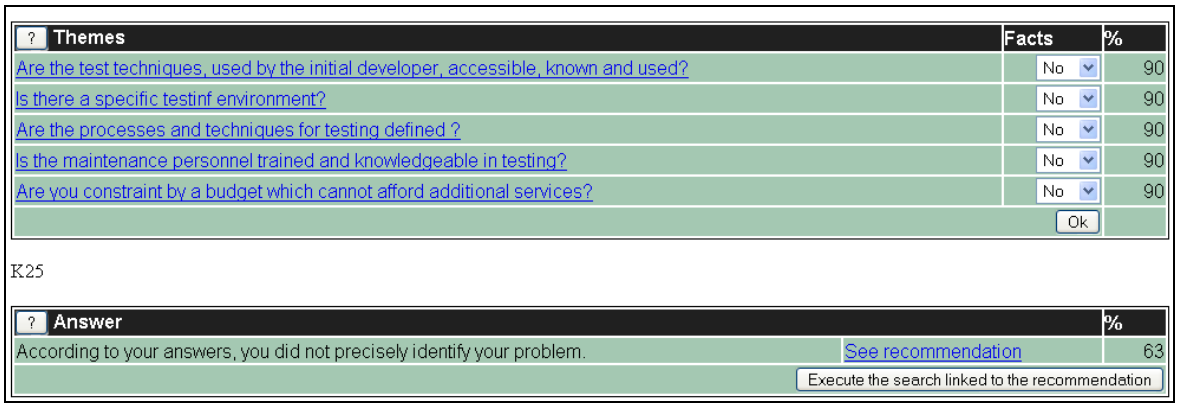


Figure 31 : Accès à une recommandation

2. Discernabilité

Ce critère est respecté par le logiciel *SM*^{Xpert}.

3. Concision

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}, en ce qui concerne la page des utilisateurs de type « expert ». En effet, l'utilisateur ne reçoit pas uniquement l'information liée à l'exécution des tâches que permet cette page. L'exemple caractéristique est la page d'accueil (Figure 24), où sont présentés une partie de la base des connaissances ainsi que l'information pour exécuter les fonctions permises. Ce qui déroute l'utilisateur de type « expert ».

4. Cohérence

Ce critère est respecté par le logiciel *SM*^{Xpert}.

5. Détectabilité

Ce critère est respecté par le logiciel *SM*^{Xpert}. Mise à part la difficulté de demander une fonctionnalité, lorsqu'une de celles-ci est voulue (comme l'ajout d'un cas problème), l'attention de l'utilisateur est dirigée vers l'information demandée.

6. Lisibilité

Ce critère est respecté par le logiciel *SM*^{Xpert}.

7. Compréhensibilité

Sur la page des utilisateurs de types « user », l'information permettant de naviguer dans la base de connaissance y est présentée. Mais il est difficile de deviner, lors d'une première visite de cette page, que cette information permet cette fonctionnalité.

Dans chaque sous-menu de la page des utilisateurs de types « user », il y a une colonne affichant un pourcentage, mais pas de titre au-dessus de la colonne. L'utilisateur ne peut donc pas savoir ce que signifie ce pourcentage. On peut voir ceci sur la Figure 32.

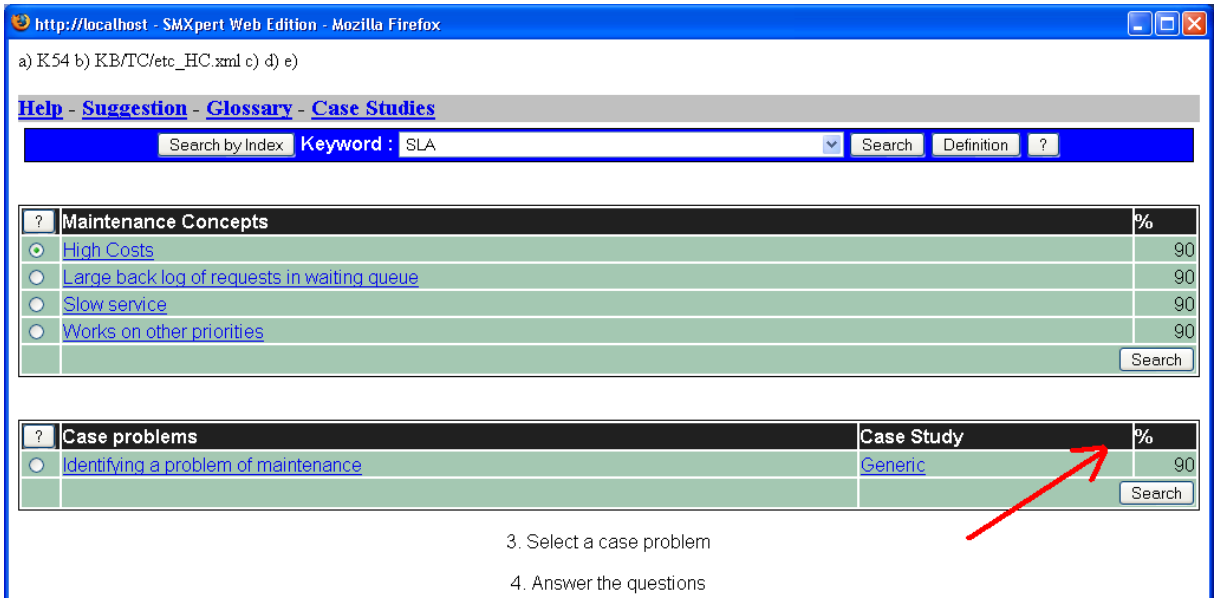


Figure 32 : Illustration du manque de compréhensibilité (a)

Le menu de la page des utilisateurs de types « expert » n'est pas également tout à fait compréhensible. Dans l'onglet « edit », il y a des fonctionnalités comme « add… », « delete… » dont on ne peut comprendre au premier abord leur utilité.

Alors que la fonction « add… » sert à l'ajout d'un cas problème.

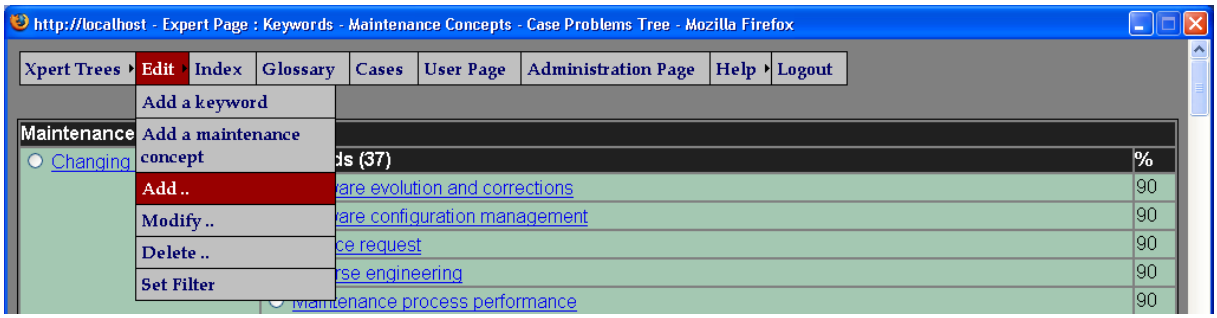


Figure 33 : Illustration du manque de compréhensibilité (b)

Adéquation des fenêtres (quand)

1. L'utilisateur surveille ou accède simultanément à plusieurs systèmes, applications ou processus.

Ce critère est rempli par le logiciel *SM*^{Xpert}, il est possible pour l'utilisateur d'accéder simultanément de manière aisée à plusieurs systèmes, applications ou processus, malgré le fait que la première page d'accueil du logiciel *SM*^{Xpert} reste toujours ouverte. Ce qui gène un peu l'utilisateur car il ne s'y attend pas.

La suivante montre que la page d'accueil du logiciel *SM*^{Xpert} reste toujours ouverte.

Figure 34 : Les multiples fenêtres de *SM*^{Xpert}

2. L'utilisateur évalue, compare ou manipule plusieurs sources d'information ou plusieurs vues d'une source unique d'information.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. En effet il est impossible de consulter plusieurs recommandations simultanément grâce à plusieurs navigations simultanées, permettant à l'utilisateur de ne pas faire séquentiellement ces mêmes navigations. Ce qui lui donnerait une vue plus large de la base de connaissance, une façon plus rapide d'obtenir une recommandation et enfin de faire moins d'effort de mémorisation des opérations déjà réalisées.

3. L'utilisateur doit avoir accès de façon occasionnelle à des composants de dialogue supplémentaires.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. Par exemple, dans la page des utilisateurs de type « expert », l'accès au glossaire permet d'obtenir l'index, tous les mots-clés, etc. Il est possible de revenir sur la page des utilisateurs de type « expert » à partir de cette page, mais le fait de cliquer sur un mot d'index pour avoir sa définition ouvre une nouvelle fenêtre avec la définition, ferme la fenêtre courante, et de cette nouvelle fenêtre il n'y a plus de possibilité d'accéder à d'autres pages, ni même revenir à la page précédente. Il faut fermer la fenêtre avec la définition et ouvrir une nouvelle session. Un exemple est illustré ci-dessous, la page du glossaire est ouverte (Figure 34) on peut voir le bouton de retour vers la page d'accueil des utilisateurs de type « expert », tandis que la Figure 35 montre le résultat du choix du mot d'index « Artefacts ».

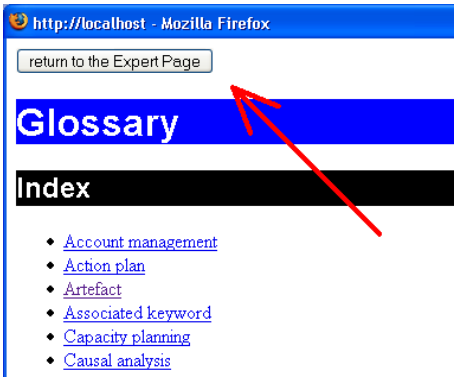


Figure 35 : Le glossaire

Figure 36 : Définition du mot Artefact

Heureusement ce cas extrême de dialogue supplémentaire n'a été détecté qu'une seule fois. Ce qui arrive lors d'un dialogue supplémentaire occasionnel (comme la demande d'une définition) est qu'il apparaît dans une nouvelle fenêtre sans lien vers la fenêtre principale qui, quant à elle, existe toujours et est cachée par la nouvelle. Ainsi pour revenir sur la fenêtre principale, il faut fermer la nouvelle fenêtre, ce qui est très perturbant.

4.2.3 Présentation de l'information 2/2

Cette section confronte le logiciel *SM*^{Xpert} aux recommandations relatives aux fenêtres, zones, groupes, etc. Tous les éléments (exemple : les tableaux) de la présentation de l'information ne sont pas repris car leurs critères ne peuvent être confrontés au logiciel *SM*^{Xpert} et pour cause, ces éléments n'existant pas dans cette version du logiciel *SM*^{Xpert}.

Recommandations relatives aux fenêtres

1. S'il est nécessaire d'afficher ou de manipuler des informations provenant de différentes sources, il convient d'envisager l'utilisation de plusieurs fenêtres ou d'une fenêtre unique contenant plusieurs zones d'entrée/sortie.

Il n'est pas nécessaire, dans l'utilisation du logiciel *SM*^{Xpert}, d'afficher ou de manipuler des informations provenant de plusieurs sources.

2. Lorsque plusieurs fenêtres sont utilisées il faut fournir une identification unique.

Ce critère est respecté par le logiciel *SM*^{Xpert}. Toutes les pages sont nommées.

3. Concevoir les positions et les dimensions des fenêtres par défaut de manière à minimiser le nombre d'opérations que les utilisateurs doivent exécuter pour accomplir une tâche.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}.

En effet, lors de la progression de la navigation dans la page des utilisateurs de type « user », les sous-menus se disposent progressivement l'un en dessous de l'autre si bien que le dernier sous-menu oblige l'utilisateur à se servir de la barre de défilement, sinon il ne le voit pas. Alors qu'il y aurait moyen d'agencer toute l'information de manière à ce que tout puisse

se faire sans utiliser la barre de défilement. La Figure 36 illustre ce fait fort contraignant de la navigation dans la base des connaissances.

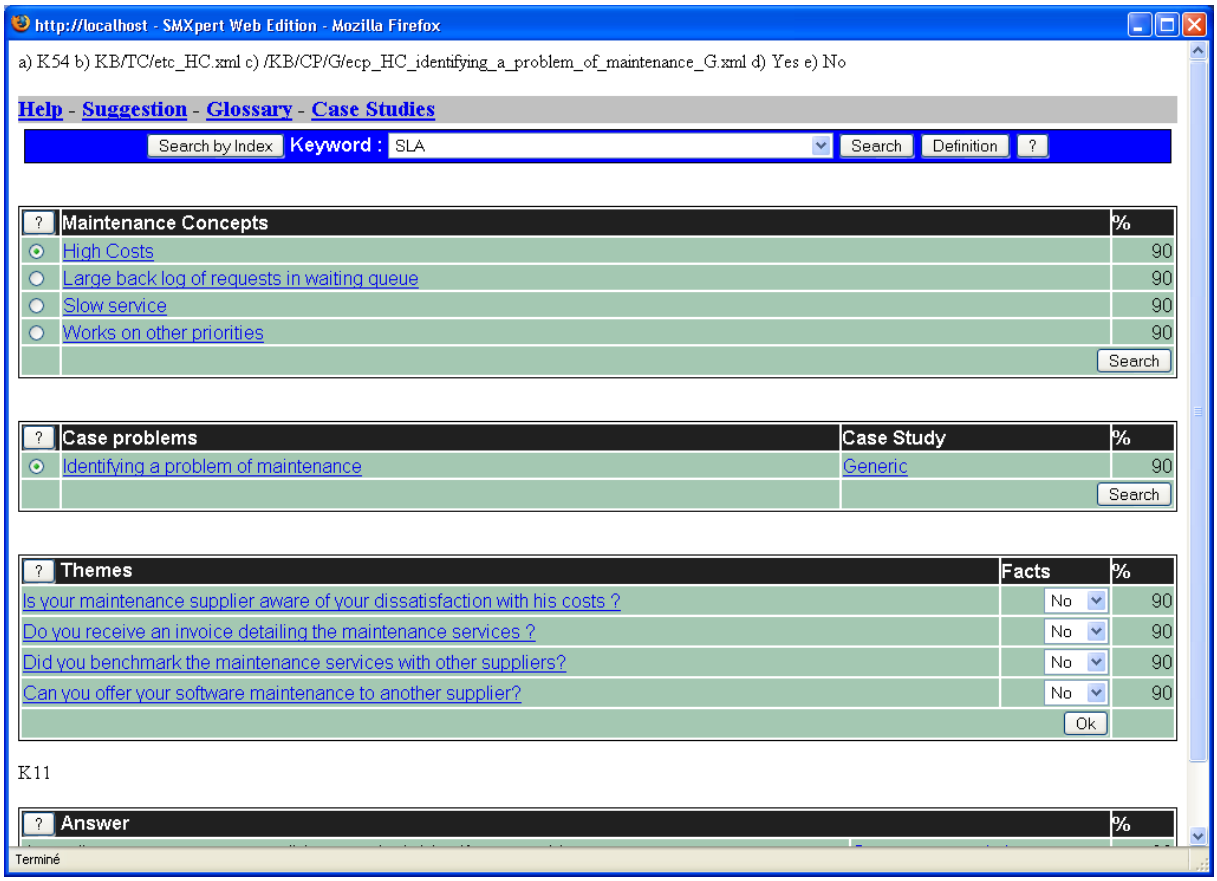


Figure 37 : page complète d'une navigation dans la base des connaissances

Il en est de même lors de l'entrée d'un cas. Chaque page du processus, trop grande pour l'écran, oblige l'utilisateur à se servir de la barre de défilement.

En règle générale, les informations sont disposées les unes en dessous des autres, aucune disposition n'a été conçue pour que toute l'information puisse apparaître à l'écran. Ce qui pose énormément problème quand intervient la liste des mots-clés, car elle allonge considérablement une page (exemple : Figure prise pour le point Concision), de même pour les mots d'index.

4. Lorsqu'une relation de subordination intervient entre fenêtres, les relations entre fenêtres principales et ses secondaires doivent toujours être visuellement apparentes.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}.

Par exemple, lorsque dans la page des utilisateurs de type « expert » on clique sur un concept de maintenance afin d'avoir sa définition, cette dernière apparaît dans une nouvelle fenêtre qui cache la fenêtre principale. Ce problème ne concerne pas uniquement les concepts de maintenance mais tous les éléments de l'ontologie du logiciel (mot-clé, index, etc.) de la page des utilisateurs de type « expert », ayant une définition accessible de la même façon que les concepts de maintenance.

5. Il convient de pouvoir discerner visuellement les éléments de commande de fenêtres exécutant différentes fonctions.

Le logiciel *SM*^{Xpert} n'a pas de fenêtre propre. Les seules fenêtres sont celles de systèmes d'exploitation qui permettent d'afficher le logiciel.

6. Placer les éléments de commande de fenêtres exécutant différentes fonctions de manière cohérente et au même endroit dans chaque fenêtre.

Etant donné que le logiciel *SM*^{Xpert} n'a pas de fenêtre propre, il ne peut respecter ce critère.

7. Si cela est approprié à la tâche, il convient d'autoriser les utilisateurs à choisir leur format de fenêtrage préféré et à le sauvegarder comme format « par défaut »

Malgré que logiciel *SM*^{Xpert} n'ait pas de fenêtre propre, que les seules fenêtres sont celles du système d'exploitation qui permettent d'afficher le logiciel, il ne respecte pas ce critère. Il n'est pas possible d'exiger que le logiciel soit affiché dans une fenêtre d'une dimension différente.

Recommandations relatives aux zones

1. Les zones sont à placer de manière homogène.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

Les zones utilisées pour afficher l'ontologie du logiciel, sont placées de manière homogène et les relations de subordination sont toujours respectées et affichées de la même façon (exemple : Figure 24).

Les zones permettant d'entrer des données sont aussi placées de manière homogène ; c'est le cas pour l'entrée d'un cas problème.

2. Le densité de l'information affichée doit être telle qu'elle ne soit pas être perçue par l'utilisateur comme trop encombrée.

Ce critère n'est pas respecté par le logiciel. Par exemple, lorsque dans une zone sont affichés les mots-clés, ces derniers sont disposés en liste (exemple : Figure 24), ce qui, vu leur grand nombre, contraint l'utilisateur à utiliser la barre de défilement pour aller du premier au dernier mot-clé et donne l'impression de désordre.

3. Si l'utilisateur doit discerner des relations entre des ensembles d'information affichés séparément, il est souhaitable d'afficher les deux ensembles d'information sur un seul écran.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. En effet, sur la page des utilisateurs de type « expert » sont affichés les concepts de maintenance, leurs mots-clés et cas problèmes associés. La Figure 37 montre qu'il y a un lien entre les concepts de maintenance, mots-clés et cas problème. Il est à noter que les 37 mots-clés ne sont pas affichés.

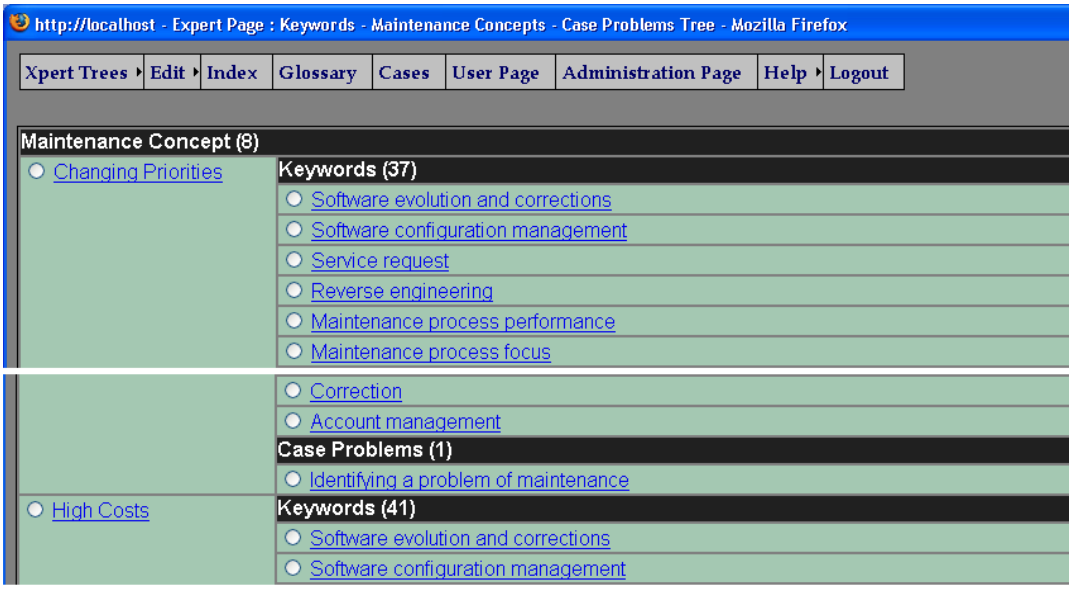


Figure 38 : Illustration du lien entre concepts de maintenance, mots-clés et cas problèmes

Mais dans une autre page sont affichés aussi (Figure 38) les concepts de maintenance avec les thèmes (questions) associés. Cette page s'affiche lorsque l'utilisateur clique sur l'onglet « Xpert Trees » et en sélectionne "maintenance concept - questions - facts tress".

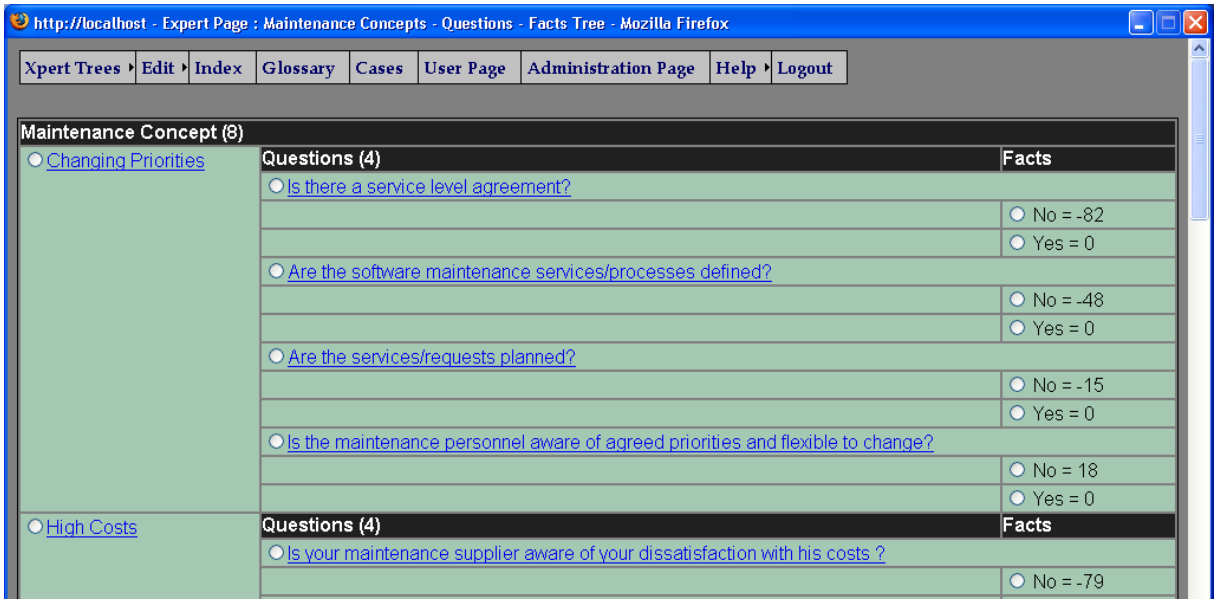


Figure 39 : Illustration du lien entre concepts de maintenance et thèmes

Il est aussi possible d'afficher les cas problèmes associés aux concepts de maintenance et les recommandations associées aux cas problèmes dans une seule fenêtre, en cliquant sur l'onglet « Xpert Trees » et en sélectionnant « Case Problems – Recommandations Tree » (Figure 39).

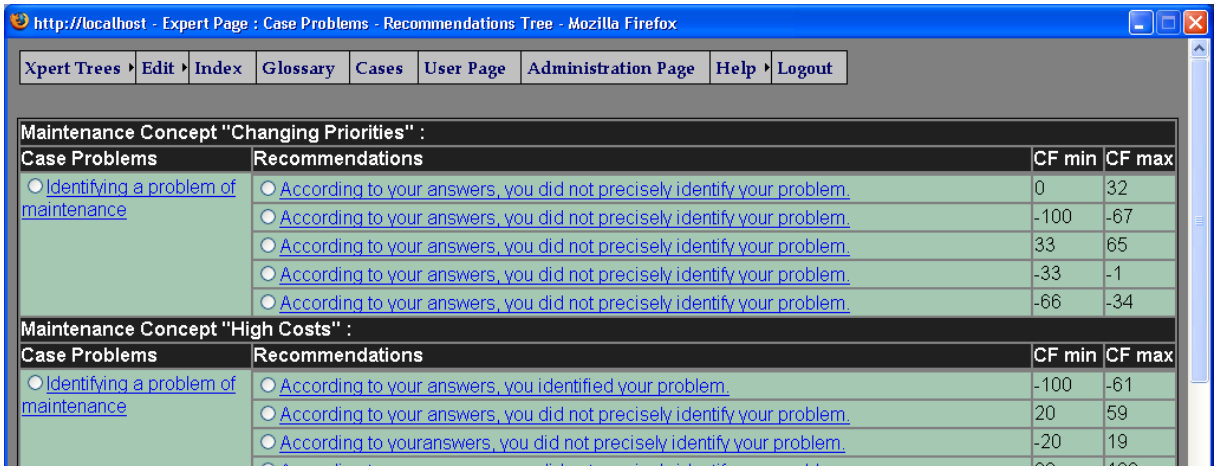


Figure 40 : Illustration de la relation entre cas problème et recommandation

Les éléments de ces trois fenêtres ont des liens entre eux, mais il n'est pas possible d'afficher tous les éléments, ou du moins l'arborescence les liant, sur un seul écran.

4. Il convient, dans la mesure du possible, d'afficher dans la zone d'entrée/sortie toutes les informations exigées pour exécuter une tâche donnée.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. Pour la procédure d'entrée d'un cas, il y a plusieurs zones d'entrée/sortie pour l'exécution de cette même tâche. Bien que cette tâche soit longue et se déroule en plusieurs étapes, la division en zones est complètement aléatoire. Lors de la première étape, il y a une seule zone d'entrée/sortie, avec des champs permettant d'insérer le nom d'un cas problème, un pourcentage, le contenu du problème et sa définition. Ensuite, lors de la deuxième étape, il y a deux zones avec des cases à cocher. Une première zone sert à associer le cas problème à un cas d'étude et l'autre à associer le cas problème à un ou plusieurs mots-clés.

Il est en effet possible de mettre toutes ces zones en une seule.

Après ces deux étapes, il y a aussi deux zones. Mais pas d'entrée/sortie. L'une d'entre elles contient les recommandations faites lors de cette procédure (donc vide la première au premier affichage), la deuxième contenant un bouton pour ajouter une recommandation.

La même critique, déjà formulée pour les deux premières étapes, peut être faite pour les zones des deux sous-étapes de l'ajout d'une recommandation.

Recommandations relatives aux groupes

1. Il convient que les groupes soient perçus comme distincts de par l'espacement et l'emplacement.

Ce critère est respecté par le logiciel *SM*^{Xpert}. Il n'y a pas beaucoup d'utilisation de groupe dans le logiciel. Il est possible d'en voir dans la procédure d'entrée d'un cas, ils sont dans les différentes zones que contient chaque étape. Et ces derniers respectent le critère.

C'est aussi le cas sur la page principale des utilisateurs de types « expert » ; les différents éléments de l'ontologie du logiciel *SM*^{Xpert} sont mis dans des groupes et il est facilement possible de voir la distinction entre eux.

Recommandations relatives aux listes

1. Les éléments et les groupes d'éléments d'une liste doivent être visuellement distincts afin de faciliter le balayage visuel.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

Les listes sont utilisées pour afficher l'ontologie du logiciel. Dans les pages des utilisateurs de type « expert » comme dans la page des utilisateurs de type « user », elles sont mises dans des zones ; ainsi cette méthode d'affichage respecte ce critère.

2. Le format des listes d'informations alphabétiques doit dépendre des conventions linguistiques (exemple : justifier à gauche les listes verticales d'informations alphabétiques pour les langues se lisant de gauche à droite).

Ce critère est respecté par le logiciel *SM*^{Xpert}.

3. Il faut justifier les informations numériques sans signe décimal (virgule ou point) à droite.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

4. Il faut justifier les informations numériques contenant des signes décimaux par rapport au signe décimal.

Il n'y a pas d'information numérique contenant des signes décimaux dans le logiciel *SM*^{Xpert}.

5. Dans les listes numériques, il convient d'utiliser une taille de police fixe avec un espacement constant.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

6. Si une liste s'étend au-delà de la zone d'affichage disponible, il convient de fournir une indication de suite de liste.

Ce critère n'est pas respecté par le logiciel *SM*^{Xpert}. Pour toutes les listes s'étendant au-delà de la zone d'affichage disponible, il n'est pas fourni d'indication de suite de liste (exemple : Figure 24).

Recommandations relatives aux champs

1. Il convient tout d'abord de bien faire la distinction visuelle entre les champs de saisie et les champs en lecture seule.

Ce critère est respecté par le logiciel *SM*^{Xpert}.

2. Si la tâche l'exige, il convient de pouvoir distinguer l'information saisie par l'utilisateur des informations générées par le système dans les champs de saisie.

Il n'existe aucune tâche exigeant de pouvoir distinguer l'information saisie par l'utilisateur de celle générée par le système ; néanmoins il est toujours possible de faire cette distinction.

4.3 Réflexion sur le logiciel SM^{Xpert}

4.3.1 Introduction

Cette section n'a pas l'intention de proposer une solution pour tous les problèmes cités dans la section précédente, mais plutôt de suggérer une esquisse de solutions. Il est aussi important d'identifier les problèmes du logiciel *SM*^{Xpert} pour déterminer les changements à investiguer lors de l'étape subséquente de cette recherche.

4.3.2 Problèmes majeurs d'interface.

4.3.2.1 Esquisse de solutions

En ce qui concerne l'interface de la page des utilisateurs de type « user », il serait intéressant de disposer les sous-menus de façon à ce que l'utilisateur ait le moins de manipulation de la barre de défilement possible.

Il serait intéressant également d'offrir à l'utilisateur de garder la trace des étapes de sa navigation courante.

Quant à l'interface des utilisateurs de type « expert », il y a plus de pain sur la planche. Tout d'abord ne plus afficher trop d'informations hétéroclites sur la page d'accueil. En se référant [10] :

« Un dialogue est considéré comme adéquat, pour une tâche, lorsqu'il permet à l'utilisateur de réaliser cette tâche de façon efficace et efficiente. » La première page de l'interface expert ne devrait donc présenter que l'information concernant les fonctions possibles via la page expert.

Il faudrait aussi séparer les boutons pour quitter la page des utilisateurs de type « expert » des boutons ayant une fonction.

Ensuite rendre les procédés de changements de la base de connaissance plus intuitifs.

4.3.3 Problèmes d'ordre conceptuels.

Certains critères, comme la facilité d'apprentissage, mettent à jour des problèmes plus importants encore que de simples problèmes d'interfaces. Il faut pouvoir comprendre les concepts sous-jacents d'un logiciel, ce qui est très difficile en utilisant SM^{Xpert}.

Par exemple l'entrée d'un cas problème correct n'a jamais été réussie, bien des questions peuvent alors se poser.

a) Premier problème

Voici pour commencer le problème le plus représentatif, qui est l'entrée d'un cas problème. Premièrement, un cas problème est relié à un cas d'étude qui relie des cas problèmes ensemble. Il faut d'abord en entrer un avant de faire la demande d'entrée d'un cas problème. Il n'est pas possible d'en insérer un pendant l'insertion d'un cas problème. Ceci doit être fait avant et donc connu au préalable.

Une fois fait, il faut aller sur l'interface des utilisateurs de type « expert », cliquer sur un concept de maintenance, puis aller sur le menu dans l'onglet « edit » et cliquer sur add. Ceci pour relier un cas problème à un concept de maintenance.

Arrivé dans le menu d'insertion d'un cas problème, le logiciel demande, sur une première page, le nom du cas problème, un « problem identification » et un « problem content ».

La page suivante demande à quel cas d'étude est relié le cas problème et à quels mots-clés lier ce cas problème.

Ainsi on peut déjà voir qu'un cas problème est relié à des mots-clés et à un concept qui lui-même est relié à des mots-clés.

La troisième page concerne l'ajout d'une recommandation. Toujours en relation avec le cas problème. La première fenêtre de l'ajout d'une recommandation nous demande d'entrer une réponse, ainsi qu'un pourcentage maximum et minimum. La seconde fenêtre nous demande en premier une redirection qui est un mot-clé. Cela sert à rediriger l'utilisateur, s'il le désire, vers un autre mot-clé ayant un rapport avec la recommandation qu'il vient de consulter. Cette seconde fenêtre nous demande aussi de lier des mots-clés à cette recommandation et enfin de lier la recommandation à des nœuds qui sont les entrées d'index.

Si un problème survient lors d'une étape de cette opération, il faut tout recommencer.

Une fois cette opération faite, étant donné qu'un cas problème est relié à un concept de maintenance et que ces derniers ont des thèmes respectifs, il faut par l'interface relier ces thèmes aux recommandations ajoutées dans l'étape précédente.

Pour cela il faut aussi aller dans l'interface des utilisateurs de type « expert » choisir dans le menu l'onglet "expert trees" et cliquer sur "maintenance concept - questions - facts tress". Ensuite, sur cette nouvelle page, il faut cliquer sur un thème du concept de maintenance pour lequel on vient d'ajouter un cas, puis aller dans le menu sur l'onglet "edit" et cliquer sur "modify". Le programme nous propose la liste des thèmes relatifs au même concept de maintenance pour lequel on veut modifier un thème. A côté de chacun des thèmes de cette liste se trouve un bouton (les boutons servent à changer l'ordre des thèmes, si l'on veut), en ce qui concerne le thème à modifier le bouton à côté est différent des autres (il propose de laisser la question à son emplacement actuel dans la liste). En cliquant sur ce bouton, et par la même occasion faire une demande de ne pas changer l'ordre, une nouvelle page est affichée, permettant de modifier un thème et de lui ajouter une recommandation. Ensuite, il faut recommencer le même procédé pour chaque thème devant être associé à une recommandation. Cette opération aussi ne peut se réaliser qu'en une seule fois ; il faut également tout recommencer si un problème survient. Il est clair que cette opération n'est pas aisée, et surtout impossible à trouver tout seul, tant les manipulations ne sont pas intuitives.

Ainsi, après plusieurs essais pour rentrer un cas, on peut résumer qu'un cas problème est associé à un ou plusieurs mots-clés et à un ou plusieurs concepts de maintenance. Ces concepts de maintenance sont reliés aussi à des mots-clés ainsi qu'à des thèmes. Les thèmes, quant à eux, sont chacun reliés à un ou plusieurs mots-clés et à une recommandation, laquelle est reliée à un cas problème.

On se rend rapidement compte qu'il y a beaucoup d'inutilités et de désordre. En outre, ce qui aurait pu être compris dans les pages des utilisateurs de type « expert » ne concorde pas exactement avec la page des utilisateurs de type « user ».

Dans la page des utilisateurs de type « user », il faut d'abord choisir un mot-clé ; le logiciel propose alors des concepts de maintenance reliés à ce mot-clé. Après avoir choisi un concept de maintenance, il faut choisir un cas problème, ensuite le logiciel affiche les thèmes auxquels il faut répondre pour avoir une recommandation.

a) Second problème

Les thèmes sont toujours posés en liste, et non un par un. De plus, le premier thème mène à une recommandation ne comportant aucune redirection vers d'autres recherches, seules les autres thèmes sont susceptibles de fournir une recommandation avec une redirection.

a) Troisième problème

Lorsque deux mots-clés sont associés à un même concept de maintenance, les cas problèmes associés à l'un des deux – et uniquement à celui-là – ne doivent pas se trouver dans les cas problèmes du second. Exemple : quand on a deux cas problèmes associés au concept de maintenance « High cost », que le premier est relié au mot-clé Facturation et le second à SLA, alors, si un utilisateur choisit, lors de sa navigation, le mot-clé « SLA », puis le concept de maintenance High cost, les cas problèmes associés au concept de maintenance « High cost » et au mot-clé « SLA » ne doivent pas lui être présentés. Le choix du mot-clé doit avoir une influence de filtrage plus fin sur les cas problèmes fournis à l'utilisateur. En effet, l'utilisateur est plus susceptible de trouver le cas problème le plus représentatif du sien si le logiciel lui fournit les cas problèmes en relation avec un mot-clé et un concept de

maintenance, plutôt que tous les cas problèmes en relation avec seulement un concept de maintenance.

Sur le schéma conceptuel, cette relation n'existe pas, il est facile de dire alors que ce problème n'en est pas un ! Le logiciel respecte bien le schéma conceptuel. Mais il est demandé dans la procédure d'entrée d'un cas problème de faire cette relation. De plus, il parait logique pour la facilitation de la navigation que cette relation existe. Voilà pourquoi c'est bel et bien un problème.

4.3.4 L'impact des problèmes d'ordre conceptuels

Les problèmes d'ordre conceptuels sont les premiers à devoir être résolus. Certains ont un impact direct sur la facilité d'utilisation du logiciel. Exposons plus clairement ceci en prenant les problèmes cités ci-dessus. Prenons tout d'abord le troisième problème d'ordre conceptuel évoqué précédemment. L'utilisateur, lorsqu'il se rendra compte que les mêmes cas problèmes lui sont présentés alors qu'il avait choisis deux mots-clés différents, risque de ne pas comprendre l'utilité du choix d'un mot-clé et trouver rébarbatif le fait de devoir systématiquement en choisir un. Hélas, ce n'est pas seulement l'unique risque, l'utilisateur pourrait très bien douter de l'efficacité du choix d'un mot-clé et de ses résultats. Il en résulte alors pour lui une difficulté dans le choix d'un mot-clé.

Voyons maintenant l'impact du second problème. Le logiciel pose toujours les thèmes en liste et non l'un après l'autre. Il est très difficile de répondre à plusieurs questions en même temps, même dans la vie réelle. Il est également impossible de savoir que le premier thème mène à une recommandation ne comportant aucune redirection vers d'autres recherches. C'est donc très difficile de comprendre ce mécanisme de question se trouvant après le choix d'un cas problème.

L'impact est encore plus explicite dans la procédure d'entrée d'un cas problème ! Il est demandé à l'utilisateur d'associer un cas problème à un concept de maintenance, ensuite de le lier à un mot-clé ; si on compare avec l'interface des utilisateurs de type « user », l'utilisateur doit choisir un mot-clé et ensuite un concept de maintenance, après cela seulement il peut choisir un cas problème. Jusque-là toute la procédure d'entrée de logiciel est aisée ; mais lorsqu'il est demandé d'associer un ou plusieurs mots-clés à une recommandation, il est difficile de comprendre pourquoi.

Il est aussi impossible de deviner que pour rentrer un cas, cela se fait en deux temps : dans un premier temps, le cas problème et les recommandations ; dans un deuxième temps, les thèmes qu'il faut relier aux recommandations. Là encore l'inverse aurait été plus logique. La conception de la procédure d'entrée d'un cas est mal faite et rend la procédure incompréhensible.

Nous voyons que les problèmes conceptuels engendrent des incompréhensions.

4.3.5 En bref...

On remarque assez rapidement que beaucoup de principes ne sont pas respectés, en commençant par la page des utilisateurs de type « expert ». La clarté, la concision et l'homogénéité ne sont déjà pas respectées ; en effet, la page des utilisateurs de type expert est énorme, peu commode et présente des éléments hétérogènes (fonctions de changement de la base des connaissances, éléments de la base des connaissances, fonction de navigation vers d'autres pages du logiciel…). Des procédés ayant des buts similaires ne se réalisent pas de la même manière, ils sont peu communs et difficiles à deviner. Le manque de compte rendu rend aussi la tâche de l'utilisateur plus ardue. Enfin la page des utilisateurs de type « expert » ne

présente pas l'information liée à l'exécution des tâches que celle-ci permet d'accomplir. Ce qui va directement à l'encontre de l'adaptation de la tâche.

Les problèmes de la page des utilisateurs de type « user » sont moins importants, le problème est surtout un manque d'aisance de la navigation dans la base des connaissances.

Outre ces problèmes d'interface, le logiciel *SM*^{Xpert} affiche également des problèmes d'ordre conceptuels importants ayant un impact sur la compréhension, l'utilité de certaines demandes, l'intuitivité et la cohérence du contenu du logiciel.

4.4 Conclusion

Comme nous le savons l'adaptation de la tâche, le caractère auto descriptif, le contrôle utilisateur, la conformité aux attentes de l'utilisateur, la tolérance à l'erreur, l'aptitude à l'individualisation et la facilité d'apprentissage sont des principes de dialogue devant être respectés afin d'avoir un logiciel intuitif, facile d'utilisation et permettant de réaliser des tâches de manière efficiente et efficace. Nous savons également que respecter les recommandations sur les objets graphiques en les concevant de manière cohérente et faciles d'utilisation ou capables d'afficher l'information pour permettre à l'utilisateur une lecture rapide et efficace, améliore aussi une interface.

Le savoir précieux que nous apportent les principes de dialogue et les recommandations ont permis une critique du logiciel et mis en avant des problèmes d'interface importants, comme la clarté, la concision et l'homogénéité. Mais mis à part ces sérieux problèmes, nous avons remarqué que peu de principes sont respectés et que peu de recommandations sont suivies.

Résolus, ces problèmes amélioreraient la facilité d'utilisation de l'interface du logiciel *SM*^{Xpert}.

Enfin, cette analyse a permis de mettre en avant des problèmes d'une autre nature, ne pouvant pas être résolus en changeant les interfaces. En effet, des problèmes d'ordre conceptuels font également défaut et rendent l'interface difficile d'utilisation de par leur impact. Les problèmes de la page des utilisateurs de type « expert » avec la procédure d'entrée d'un cas, ou encore les problèmes concernant les thèmes en sont une bonne illustration.

Nous voyons donc que pour améliorer le logiciel *SM*^{Xpert} il faut des changements importants.

Chapitre 5 Proposition de ré-ingénierie

5.1 Introduction

Nous avons conclu au chapitre précédent que beaucoup de principes d'interface n'étaient pas respectés, que les problèmes de la page des utilisateurs de type « user » étaient moins importants que ceux de la page des utilisateurs de type « expert ». Nous avons également mis en avant d'autres problèmes d'ordre conceptuel et finalement terminé sur la nécessité d'un changement important du logiciel *SM*^{Xpert}.

Ce chapitre présente les changements à opérer, leur raison d'être, et apporte une proposition de ré-ingénierie sur base de ces changements.

5.2 Analyse des changements à opérer

Après avoir analysé les faiblesses du logiciel *SM*^{Xpert}, nous avons conclu qu'il fallait améliorer le logiciel *SM*^{Xpert}. Nous allons voir les changements nécessaires à apporter, ceux

qui ont pour but d'améliorer les faiblesses mises à jour au chapitre précédent, mais également d'autres qui permettent d'améliorer encore le logiciel.

5.2.1 Changements directement liés à l'analyse des faiblesses

Il y a tout d'abord une volonté d'améliorer les interfaces. Mais nous avons également vu que des changements plus importants encore devaient être faits. Ainsi quatre problèmes majeurs ont été identifiés au chapitre précédent :

1. Améliorer les interfaces en conformité avec les principes généraux sur la présentation de l'information. Il y a la page des utilisateurs de type « expert » inutilement énorme, le manque d'homogénéité, etc. Il faut aussi améliorer la compréhension du logiciel en respectant le caractère auto descriptif. La page des utilisateurs de type « user » doit aussi subir quelques modifications pour faciliter la navigation et permettre de se repérer dans la base des connaissances (exemple : que l'utilisateur puisse voir le mot-clé et le concept de maintenance qu'il a choisis lorsqu'il sélectionne un cas problème).
2. La procédure d'entrée d'un cas contient beaucoup d'inutilités, de désordre, elle est source d'incompréhensions et elle ne donne pas une représentation du logiciel concordant avec la représentation de ce même logiciel, laissée par la page des utilisateurs de type « user ». Il faut que la procédure d'entrée d'un cas soit plus intuitive, qu'elle permette une représentation du logiciel égale à celle donnée par l'interface des utilisateurs de type expert. C'est réalisable en respectant le schéma conceptuel. Il faut donc modifier la procédure d'entrée d'un cas afin qu'elle respecte le schéma conceptuel. Et surtout que les étapes s'agencent de la même manière qu'un déroulement d'utilisation de la page des utilisateurs de type « user ». L'intuitivité sera nettement plus accrue.
3. Les thèmes sont toujours posés en liste, et non un par un. Et le premier thème mène à une recommandation ne comportant aucune redirection vers d'autres recherches.
4. Les cas problèmes sont associés uniquement aux concepts de maintenance et sont toujours les mêmes, quel que soit le mot-clé choisi. Il faut donc changer le logiciel pour que les cas problèmes soient associés aux concepts de maintenance et également aux mots-clés.

5.2.2 Autres changements

Il est impératif que les problèmes cités ci-dessus soient résolus. Mais il y a encore d'autres volontés de changements indirectement mis à jour par l'analyse des faiblesses lors de concertations avec des utilisateurs de SM^{Xpert}, dont un expert en maintenance. Voici ces propositions :

1. Ajout d'une fonctionnalité concernant la page des utilisateurs de type « user ». Il a été demandé d'ajouter une sorte de navigateur affichant, tout au long de l'utilisation de cette page, l'endroit où se trouve l'utilisateur dans l'arborescence de la base des connaissances. Ce navigateur afficherait ainsi à l'utilisateur le chemin déjà parcouru dans la bases des connaissances (exemple : mot-clé et concept de maintenance choisis) et également les chemins qui lui sont accessibles. L'information affichée sur le navigateur serait des liens hypertextes, facilitant davantage la navigation et l'accélérant.

2. Remplacer la technologie permettant de mémoriser la base des connaissances. Le logiciel SM^{Xpert} mémorise sa base des connaissances, et autres données de gestion, par

l'intermédiaire de fichiers XML. Ce qui est demandé est de remplacer les fichiers XLM par un système de gestion de base de données.

3. Modifier le déroulement de la navigation dans la page des utilisateurs de type « user ». *SM*^{Xpert} a pour but d'améliorer le processus de maintenance dans une organisation de maintenance. C'est un logiciel qui doit connaître les problèmes des ingénieurs de la maintenance et proposer des solutions à ces problèmes. La suggestion faite est un logiciel qui se comporterait comme un 'médecin', c'est-à-dire qu'il poserait une ou plusieurs questions aux utilisateurs afin de déterminer leur problème et de proposer une solution. La section 5.3.3 explique en détail cette proposition de modification.

Nous avons donc sept exigences en vue d'améliorer *SM*^{Xpert}.

5.3 Nature des changements envisagés : Ré-ingénierie ou restructuration ?

A présent que nous avons mis le doigt sur les changements à opérer, nous allons voir quel processus est nécessaire pour effectuer ces modifications. Nous avons vu deux processus de changement majeur : la restructuration et le ré-ingénierie.

5.3.1 Hypothèse de restructuration

Analysons premièrement si les changements à opérer constituent de la restructuration. Chikofsky et Cross [5] nous indiquent premièrement que c'est « la transformation d'une forme de représentation vers une autre au même niveau d'abstraction, tout en préservant le comportement externe du système. » (A) Deuxièmement, que cela peut être le passage d'un système non structuré vers un système structuré, sans connaître le contenu ni même le but (B). Troisièmement, maintenir par prévention d'amélioration de l'état physique d'un système en prenant de nouveau standard (C). Enfin, ajuster un système pour répondre par la suite à de nouvelles contraintes ne nécessitant pas de réévaluation d'un niveau d'abstraction plus élevé (D). A présent, regardons point par point si les sept changements peuvent constituer de la restructuration.

A. Les sept changements constituent-ils une « transformation d'une forme de représentation vers une autre au même niveau d'abstraction, tout en préservant le comportement externe du système. » ? [5]

1. **Améliorer les interfaces**. Les demandes de changement d'interface ne modifient pas le comportement externe du système. Et ce n'est pas non plus un changement d'une forme de représentation.
2. **Changement de la procédure d'entrée d'un cas**. Cela change le comportement externe du système.
3. **Changement de l'agencement des thèmes et des recommandations qu'ils fournissent**. Le comportement externe du programme se voit également modifié par un changement n'étant pas une transformation de la forme de représentation.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ce changement entraîne la même remarque que pour les deux précédents.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Un ajout ne constitue pas un changement de forme de représentation ; de plus, le comportement externe du système ne sera peut-être pas modifié dans le

sens où les résultats seront différents, mais dans le sens où il améliorera sa possibilité d'interaction.

6. **Changement des technologies de stockage de données, passage de fichier XML vers un système de gestion de base de données**. Ce changement est une transformation de la forme de représentation, celle de la mémorisation des données, sans changer le comportement externe du système. On peut donc considérer que ce changement constitue une restructuration dans le sens de la définition de Chikofsky et Cross.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Pour ce changement, on peut à nouveau faire la même remarque que pour les changements numéros 2, 3 et 4.

B. Tous les changements seraient-ils une transformation code vers code, pour le passage d'une système non structuré vers un système structuré, sans connaître le but du logiciel ?

1. **Améliorer les interfaces**. Il est possible de changer une interface et de la rendre conforme à une norme sans pour autant connaître le but du logiciel.
2. **Changement de la procédure d'entrée d'un cas**. La rendre plus simple et plus compréhensible. Pour pouvoir opérer un tel changement il est nécessaire de connaître la procédure, les informations qu'elle doit mémoriser et son but.
3. **Changement de l'agencement des thèmes et des recommandations qu'ils fournissent**. Il est clair que ce changement requiert une redéfinition des relations existantes entre les thèmes eux-mêmes, les cas problèmes et les recommandations. La structure du code permettant d'afficher les thèmes ne sera pas changée mais bien le code lui-même. C'est un changement du comportement du système, cela ne peut se réaliser sans connaître son but.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ce changement est de nature très similaire au précédent ; dans ce cas-ci, les résultats attendus diffèrent également.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Cette demande ne peut pas être réalisée dans le cadre d'une restructuration puisque la fonctionnalité n'existe pas encore.
6. **Changement des technologies de stockage des données, passage de fichier XML vers un système de gestion de base de données**. Cette modification peut être réalisée sans connaître le programme ni son but.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Ce changement est également, comme les changements numéros 2 et 3, un changement du code avec pour résultat un comportement différent du système.

C. Est-il possible de considérer les demandes de changement comme une restructuration ayant pour but de la maintenance préventive ?

1. **Améliorer les interfaces**. On peut facilement voir les demandes de changement d'interface comme des requêtes de correction, mais réactive et non proactive. En effet, la volonté de changement d'interface est une volonté des utilisateurs après sortie du logiciel ; elle peut être vue comme un rapport d'erreur. C'est donc de la maintenance corrective.
2. **Changement de la procédure d'entrée d'un cas**. Ceci pourrait être vu comme une requête de modification formulée après manifestation d'impossibilité d'utilisation. Ce n'est donc pas de la maintenance préventive mais corrective.

3. **Changement de l'agencement des thèmes et des recommandations qu'ils fournissent**. Cette demande constitue un changement des exigences, c'est donc de la maintenance perfective.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Comme pour le second changement, cela constitue de la maintenance perfective car c'est un changement des exigences.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Cette demande de changement est en réalité une nouvelle fonctionnalité, c'est donc de la maintenance adaptative.
6. **Changement des technologies de stockage de données, passage de fichier XML vers un système de gestion de base de données**. Cette demande serait plutôt vue, dans le processus de la maintenance, comme une migration ; c'est un changement de plate-forme, sans pour autant subir de modifications fonctionnelles.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Cette demande de changement est une requête modifiant aussi les exigences ; elle constitue alors de la maintenance perfective.

D. Les changements voulus seraient-ils de la restructuration pour un ajustement du système afin qu'il puisse répondre à de nouvelles contraintes dans le futur, mais sans une réévaluation d'un niveau d'abstraction plus élevé ?

1. **Améliorer les interfaces**. Ce n'est pas le cas pour les changements d'interface car il ne s'agit pas du tout d'un ajustement du système en vue de contraintes futures.
2. **Changement de la procédure d'entrée d'un cas**. Le changement de procédure d'entrée d'un cas a pour but de faciliter cette dernière. Mais il est possible de voir cette demande de changement comme un ajustement du système afin de répondre à de nouvelles contraintes. Plus simple, cette dernière serait effectivement plus facile à changer en cas de modifications des informations constituant un cas.
3. **Changement de l'agencement des thèmes et des recommandations qu'ils fournissent**. Ceci est une rectification conceptuelle ; quelles que soient les éventuelles contraintes futures, il est impératif d'effectuer ces changements.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ce changement. Ceci est également une rectification conceptuelle.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Une amélioration ne constituant pas un ajustement, cette demande n'est donc pas un ajustement en vue d'éventuelles contraintes futures.
6. **Changement des technologies de stockage de données, passage de fichier XML vers un système de gestion de base de données**. Cette demande constitue un ajustement en vue de contraintes futures. En effet, dans un système de gestion de base de données, la gestion des données est beaucoup plus aisée (ajout, mise à jour, suppression …). Si à l'avenir de nouvelles données devaient être sauvegardées, il serait plus facile d'opérer les modifications nécessaires avec un système de gestion de base de données qu'avec des fichiers XML.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Cette demande n'est pas non plus un ajustement du système lui permettant d'évoluer plus facilement par la suite.

Un tableau résume cette analyse détaillée suivant chacune des restructurations possibles :

	A	B	C	D
Premier	Non	Oui	Non	Non
Second	Non	Non	Non	Oui
Troisième	Non	Non	Non	Non
Quatrième	Non	Non	Non	Non
Cinquième	Non	Non	Non	Non
Sixième	Oui	Oui	Non	Oui
Septième	Non	Non	Non	Non

De tous les types de restructuration, 4 changements ne peuvent pas être de la restructuration. Seuls le changement des interfaces, la mise en place d'un système de gestion de base de données et – de manière plus nuancée – le changement de la procédure d'entrée d'un cas peuvent en faire partie. Tous ces problèmes ne peuvent donc se régler par la restructuration.

5.3.2 Hypothèse de ré-ingénierie

A présent que nous savons que tous les changements ne pourront pas être effectués par de la restructuration, analysons-les sous l'angle de la re-ingénierie.

La définition de Chikofsky et Cross [5], plus adaptée à l'informatique est : « La re-ingénierie est l'examen et l'altération d'un système pour le reconstituer dans une nouvelle forme et en implémenter cette nouvelle forme. »

1. **Améliorer les interfaces**. Les demandes de changement d'interface ne sont pas de la re-ingénierie dans le sens où il y a bien un examen et une altération du système, mais où sa forme reste inchangée.
2. **Changement de la procédure d'entrée d'un cas**. C'est bien un examen et une altération, mais pas du système complet, uniquement d'une procédure constituant une partie de ce système. Cette demande de changement peut-être considérée comme une re-ingénierie localisée.
3. **Changement de l'agencement des thèmes et des recommandations qu'ils fournissent**. Ce changement est aussi voulu après examen, et il engendrera également une altération du système, mais dans ce cas le changement est vraiment minime et n'altère pas une procédure complète. Ce changement ne peut pas être une re-ingénierie, même localisée.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ce changement amène la même réflexion que pour le changement précédent.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Cette demande n'exige pas d'examen, il y a bien une altération du code mais pas réellement de nouvelle forme.
6. **Changement des technologies de stockage des données, passage de fichier XML vers un système de gestion de base de données**. Cette demande de changement n'affecte pas non plus la forme du système.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Cette demande nécessite plus qu'une simple explication, car son impact est énorme. C'est l'objet de la section suivante (5.3.3).

La modification du déroulement de la navigation dans la page des utilisateurs de type « user », est un changement important. Il est intéressant de préciser ce changement et ensuite de voir l'impact qu'il aura ainsi que tous les autres changements.

5.3.3 Modification du déroulement de la navigation : changement majeur

La volonté de changement la plus importante mise à jour par l'intermédiaire de concertation est le changement du déroulement de la navigation dans la page des utilisateurs de type « user ». La navigation voulue est totalement différente et jugée plus adéquate par les experts en maintenance. Lors de la navigation, l'utilisateur, après avoir choisi un mot-clé et un concept de maintenance, se voit poser un premier thème de la part du système auquel il peut répondre affirmativement ou négativement. Suivant cette réponse le système pose soit une seconde question, soit fournit une recommandation. Une fois qu'un utilisateur reçoit une recommandation, il peut consulter un cas problème. Ce cas serait en réalité une illustration, un exemple passé résolu par la recommandation.

Le logiciel SM^{Xpert} est un logiciel hybride avec une première partie du déroulement basée sur le raisonnement par cas, et lorsqu'un cas est trouvé, il est demandé de répondre à des thèmes. Cette partie est basée sur le raisonnement par règle.

La base de connaissance est principalement constituée de cas, et le moteur d'inférence sert à trouver une recommandation suivant les thèmes posés par le système. L'évolution de la base des connaissances est incrémentale par l'insertion de nouveaux cas et leur solution.

Ce type de système, avec un raisonnement particulier qui est une combinaison du raisonnement par cas et du raisonnement par règle, est jugé inadéquat pour résoudre un problème d'une organisation de maintenance. Prenons, pour illustrer, une visite chez le médecin :

> Le médecin que vous consultez cherche à savoir quel est votre problème en vous posant des questions ou en vous auscultant, ce qui revient à répondre à des questions qu'il se pose à lui-même afin d'identifier tous les symptômes et de déterminer quelle est votre maladie.
>
> Ensuite il vous fournit une recommandation (hygiène de vie ou médicaments). Mais lorsque vous entrez dans son cabinet, il ne vous demande pas d'analyser les maladies d'anciens patients et de déterminer laquelle correspond le plus à la vôtre.

Il est plus facile de résoudre un problème dans une organisation de maintenance par le type de navigation mentionné plus haut et similaire à l'illustration faite avec la visite chez le médecin qu'illustré dans l'analogie avec le médecin.

Le type de raisonnement dans l'illustration de la visite chez le médecin et dans la nouvelle navigation désirée est un raisonnement basé uniquement sur des règles. La recherche du problème se fait par déduction grâce à des règles comme :

Si … Alors …

Cette amélioration vise donc à changer le type de raisonnement du logiciel.

5.3.4 Impact des changements : ré-ingénierie

A présent que nous avons bien défini la modification du déroulement de la navigation dans la page des utilisateurs de type « user », analysons l'impact de ce changement, ainsi que l'impact des autres changements.

1. **Améliorer les interfaces**. Cela ne demande pas un grand effort ; un changement d'interface est vraiment un changement de code local, n'impliquant pas le changement d'autre partie d'un logiciel.
2. **Changement de la procédure d'entrée d'un cas**. Il faut changer l'interface, mais aussi le code gérant cette procédure pour pouvoir changer les étapes afin que leur agencement soit plus logique. Une partie du code sera donc changée.
3. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ce changement nécessite de revoir la relation entre concepts de maintenance, thème, recommandations et entre les thèmes eux-mêmes, c'est donc l'analyse du logiciel qui doit être modifiée pour permettre une relation différente entre ces derniers. Ensuite implémenter ces changements. L'interface ne changera pas, mais il faudra mémoriser les relations en permettant de mémoriser de nouveaux thèmes et d'indiquer les concepts de maintenance auxquels ils sont liés, quelle recommandation ils fournissent et comment tous les thèmes se suivent. Ceci doit être fait en modifiant les données exigées dans la procédure d'entrée d'un cas. Le code gérant la navigation doit être changé afin que les bons thèmes soient affichés suivant le concept de maintenance choisi et l'un après l'autre. C'est donc deux parties du code qui devront être modifiées.
4. **Présence d'une relation entre cas problème, concepts de maintenance et mots-clés**. Ceci est également un changement conceptuel dans lequel il est nécessaire de revoir l'analyse du logiciel. Ensuite changer le code gérant la relation entre cas problème, concepts de maintenance et mots-clés pour permettre le choix de cas problèmes différents lorsqu'un utilisateur choisit deux mots-clés différents pour le même concept de maintenance. Ceci se fait aussi dans le code gérant la navigation. Il faut encore changer les données exigées dans la procédure d'entrée d'un cas, car c'est dans cette procédure que doit être indiqué cette relation. Présentement ces informations sont demandées, mais il ne semble pas qu'elles soient prises en compte car on ne distingue pas les effets lors de la navigation. Il est alors difficile de déterminer quelle partie du code pose problème, soit la procédure d'entrée d'un cas, soit la navigation. Il faudra donc réviser ces deux parties de code.
5. **Amélioration de la page des utilisateurs de type « user » avec l'ajout d'une sorte de navigateur**. Il faut principalement changer les interfaces. Une petite modification du code gérant la navigation est aussi nécessaire afin de permettre l'activation d'hyper lien sur ce navigateur.
6. **Changement des technologies de stockage de données, passage de fichier XML vers un système de gestion de base de données**. L'impact, sur le logiciel, de cette modification sera le changement de la couche communiquant avec les technologies de stockage de données.
7. **Modification du déroulement de la navigation dans la page des utilisateurs de type « user »**. Ce changement est le changement le plus important. C'est le fondement du logiciel SM^{Xpert} qui est modifié. Pour passer du système hybride au seul raisonnement par règle, il nécessite un changement de tout le fonctionnement du programme. Car même la partie fondée sur le raisonnement par règle n'est plus la même. Le logiciel ne pose plus systématiquement quatre thèmes et fournit une recommandation suivant toutes les réponses. La notion de cas problème devient

différente aussi, elle n'a plus une place prédominante et est reléguée au rang d'illustration. Le cas problème n'est plus mis en relation qu'avec une recommandation. Cette modification demande à vrai dire une nouvelle analyse.

Au vu de l'analyse de l'impact de chacune des demandes de changement, on se rend compte que la dernière est la plus importante, c'est un tout autre logiciel qui est demandé avec une spécification différente. Cette demande de changement prend même le pas sur les trois premières. En effet, avec un modèle de données différent et une notion de cas problème différente également, la procédure d'entrée d'un cas problème se réduira à la mémorisation d'un cas en relation avec une recommandation. En ce qui concerne le second changement, ce ne sera plus une liste de thèmes, mais bien un seul à la fois ! On remarque par ailleurs que le second et le dernier changement sont intimement liés ; en réalité le second amorce en quelque sorte le changement le plus important. Enfin, le troisième changement est aussi pris en compte dans le nouveau type de navigation voulu. Effectuer le changement le plus important s'avère avantageux car il permet de contourner la correction de trois autres problèmes importants.

S'ajoute à cela un changement des interfaces et ensuite de la technologie de stockage des données, ce qui a pour impact un changement de la couche de communication. Il ne reste guère d'élément du logiciel SM^{Xpert} qui ne se verra pas modifié (comme par exemple le choix d'un mot-clé et ensuite d'un concept de maintenance suivant le mot-clé choisi). Par conséquent, il n'y a pas grand-chose à garder !

L'essence même de la re-ingénierie, comme nous le savons, c'est de refaire radicalement la conception d'une organisation d'un logiciel. La modification du déroulement de la navigation dans la page des utilisateurs de type « user » est un changement fondamental qui, avec les autres changements, demande une re-ingénierie du logiciel.

5.4 Conclusion

Partis de l'idée qu'il était nécessaire d'opérer des changements, nous les avons déterminés et avons ensuite analysé quel processus, entre la restructuration et la ré-ingénierie, était nécessaire pour les effectuer. Nous avons vu qu'il était impossible d'effectuer tous les changements par une restructuration. Nous nous sommes penchés alors sur la ré-ingénierie. Après analyse des six premiers changements, il était encore trop audacieux de parler de ré-ingénierie, mais l'addition du dernier changement a forcé la décision de re-ingénierie : la modification du déroulement de la navigation dans la page des utilisateurs de type « user ». En effet, cette modification change le fondement même du logiciel SM^{Xpert}. C'est le passage d'un logiciel hybride combinant le raisonnement par cas et le raisonnement par règle vers un unique raisonnement par règle.

Chapitre 6 Vers une nouvelle version

A présent que nous avons conclu qu'il fallait une ré-ingénierie du logiciel SM^{Xpert}, présentons dans ce chapitre la nouvelle conception appelée $SM^{Xpert\ v2}$ issue de cette décision de ré-ingénierie. Cette nouvelle conception n'est pas entièrement implémentée, seules les fonctionnalités importantes ont été réalisées. Ces fonctions sont exposées au lecteur dans ce chapitre.

Pour une conception de logiciel, il est important tout d'abord d'exprimer les besoins des utilisateurs, il faut que cette information soit claire. Identifier les utilisateurs et définir leurs attentes du système est un début primordial.

« Cela permet de faire le contour du système à modéliser et de capturer les fonctionnalités principales. » [14] Ceci peut être fait grâce au cas d'utilisation d'UML.

La suite de l'analyse sera de définir un schéma conceptuel du logiciel, afin de cerner les concepts importants du domaine d'application et les liens entre ces derniers.

Les cas d'utilisation et le schéma conceptuel nous offre une vue encore abstraite de ce que va traiter le logiciel, l'apport d'une modélisation du comportement du logiciel est intéressant du point de vue de la compréhension de sa dynamique.

Il a été aussi demandé d'installer un service de base de données pour stocker les données permanentes du logiciel, un diagramme entité associations de la base de données sera également fourni.

Il faudra aussi présenter l'architecture logique et physique du logiciel ; ces éléments seront présentés lors d'une comparaison de la nouvelle version et de l'ancienne version du logiciel, comparaison permettant également de distinguer concrètement les changements.

Enfin, les technologies ayant permis le développement seront aussi évoquées.

6.1 Les cas d'utilisation du logiciel SM$^{Xpert\ v2}$

Cette partie concernant les cas d'utilisation du logiciel *SM$^{Xpert\ v2}$* permet de définir correctement les besoins des utilisateurs et de percevoir les fonctionnalités et les mécanismes du système.

Cette partie commence par définir les rôles de l'utilisateur, ensuite des schémas montrent les cas d'application que peut faire chaque utilisateur. Pour de plus amples explications des schémas, le lecteur est amené à lire l'annexe 2.

Définitions

- *redirection* : mécanisme permettant au système d'insérer un mot-clé à la place de l'utilisateur pour le diriger dans une nouvelle navigation.
- *se logger* : entrer son nom d'utilisateur dans le logiciel afin que celui-ci reconnaisse la personne qui désire accéder à ses services.
- *pages des utilisateurs de type « user »* : pages permettant la navigation dans la base de connaissance.
- *pages des utilisateurs de type « expert »* : pages permettant d'opérer des changements dans la base de connaissance.
- *pages des utilisateurs de type « administrateurs »* : page permettant la gestion des utilisateurs ainsi que leur profil et statut.
- *Profil d'utilisateur* : 4 critères constituent le profil d'un utilisateur. Ces critères sont le niveau de maturité, le type d'utilisateur (utilisateur/client ou manager/programmeur), la place (externe, interne) et la taille de son organisation.

Identification et rôles des utilisateurs

- Administrateur
- User (chaque utilisateur ayant un certain niveau de connaissance).
- Expert

L'administrateur a le droit de gestion des utilisateurs ainsi que de leur profil et statut.

L'administrateur a accès aux pages permettant d'opérer des changements dans la base des connaissances ainsi que naviguer dans la base des connaissances.

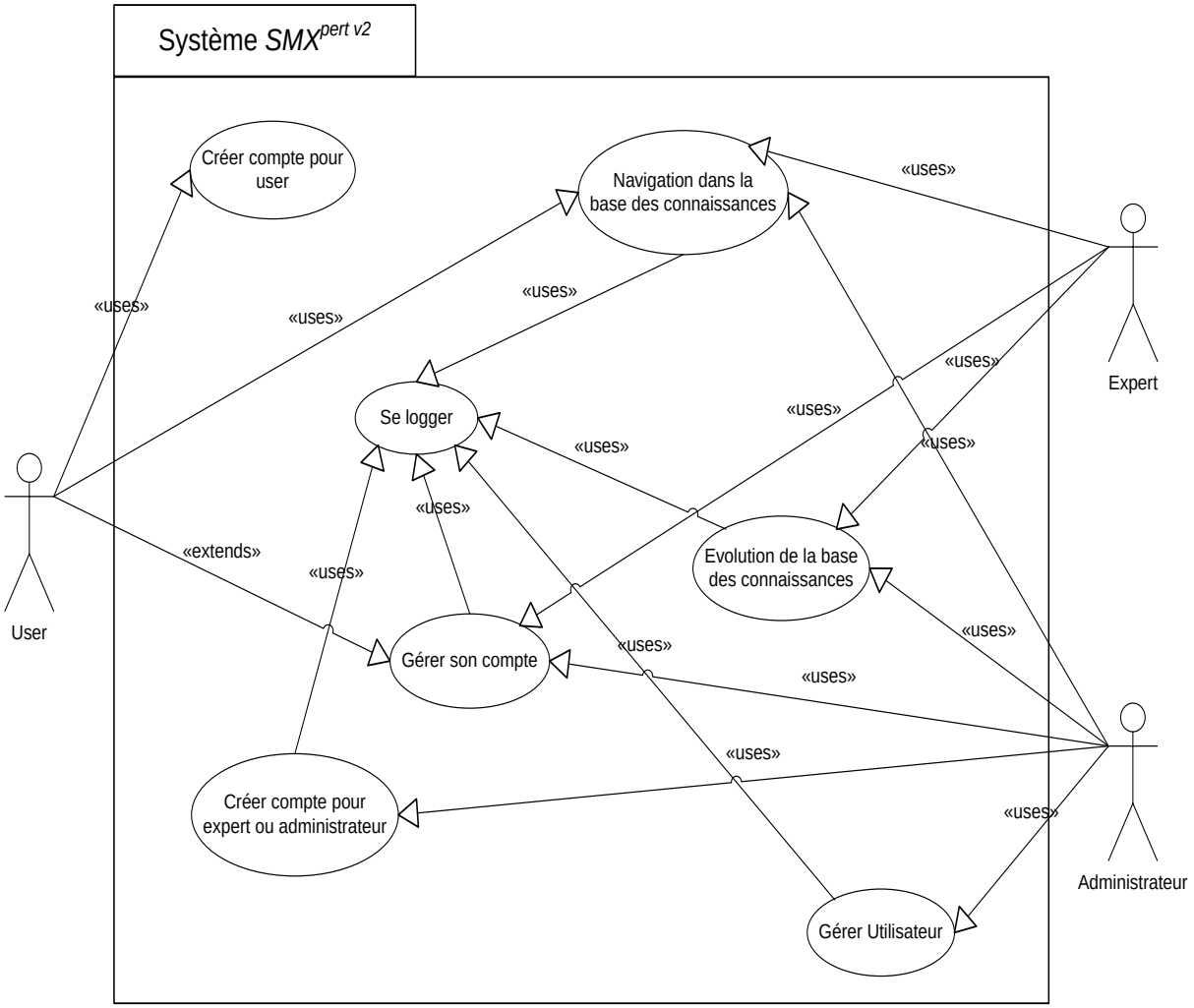
Un compte d'utilisateur de type administrateur ne peut être créé que par un utilisateur de type administrateur.

Lorsqu'il se logge sur le système, ce dernier est amené automatiquement vers la page d'accueil des pages permettant la gestion des utilisateurs ainsi que de leur profil et statut.

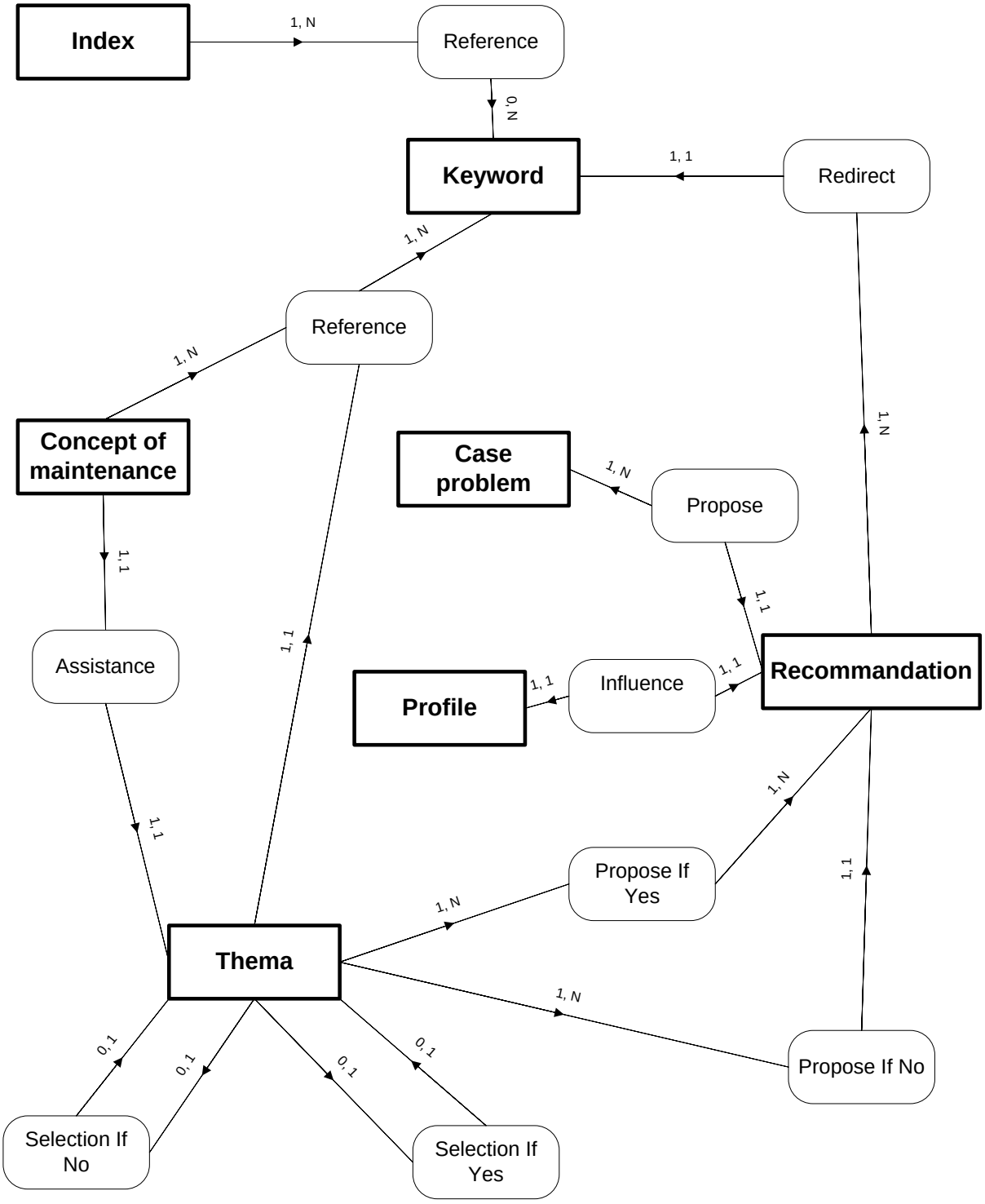
L’utilisateur a le droit de naviguer dans la base de connaissance.

L’expert a le droit d’opérer des changements dans la base de connaissance.
L’expert a accès aux pages de navigation dans la base de connaissance.
Un compte d’utilisateur de type expert ne peut être créé que par un utilisateur de type administrateur.
Lorsqu’il se logge sur le système, ce dernier est amené automatiquement vers la page d’accueil des pages permettant de modifier la base de connaissance.

Cas d’utilisation du logiciel *SMXpert v2*

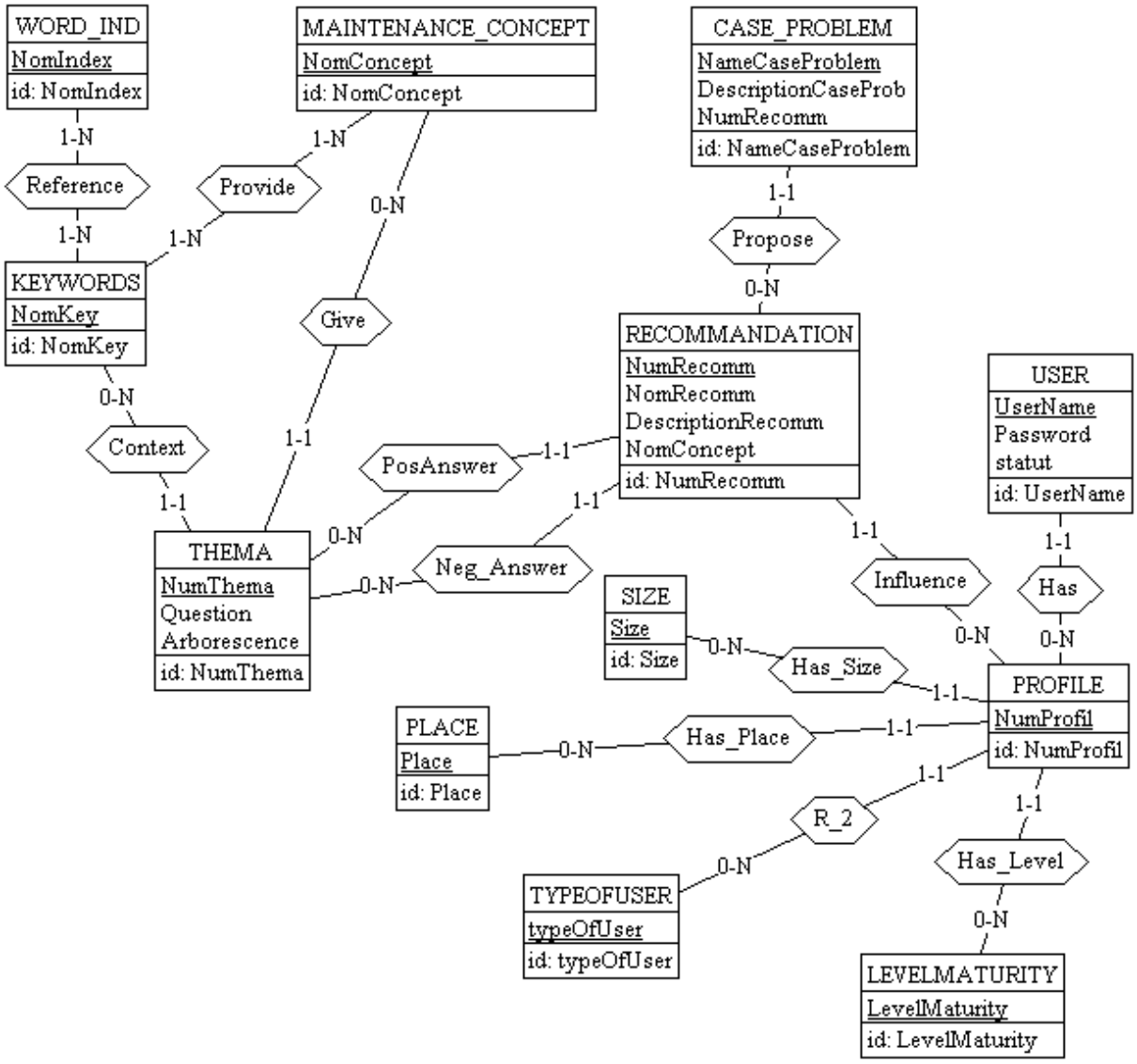


6.2 Schéma conceptuel du logiciel SM^{Xpert}



Thema.Selection_If_No ∩ Propose_If_No = Φ
Thema.Selection_If_Yes ∩ Propose_If_Yes = Φ

6.3 Modélisation statique : Diagramme entité association de la base de données



6.4 Modélisation du comportement du logiciel SM^{Xpert}

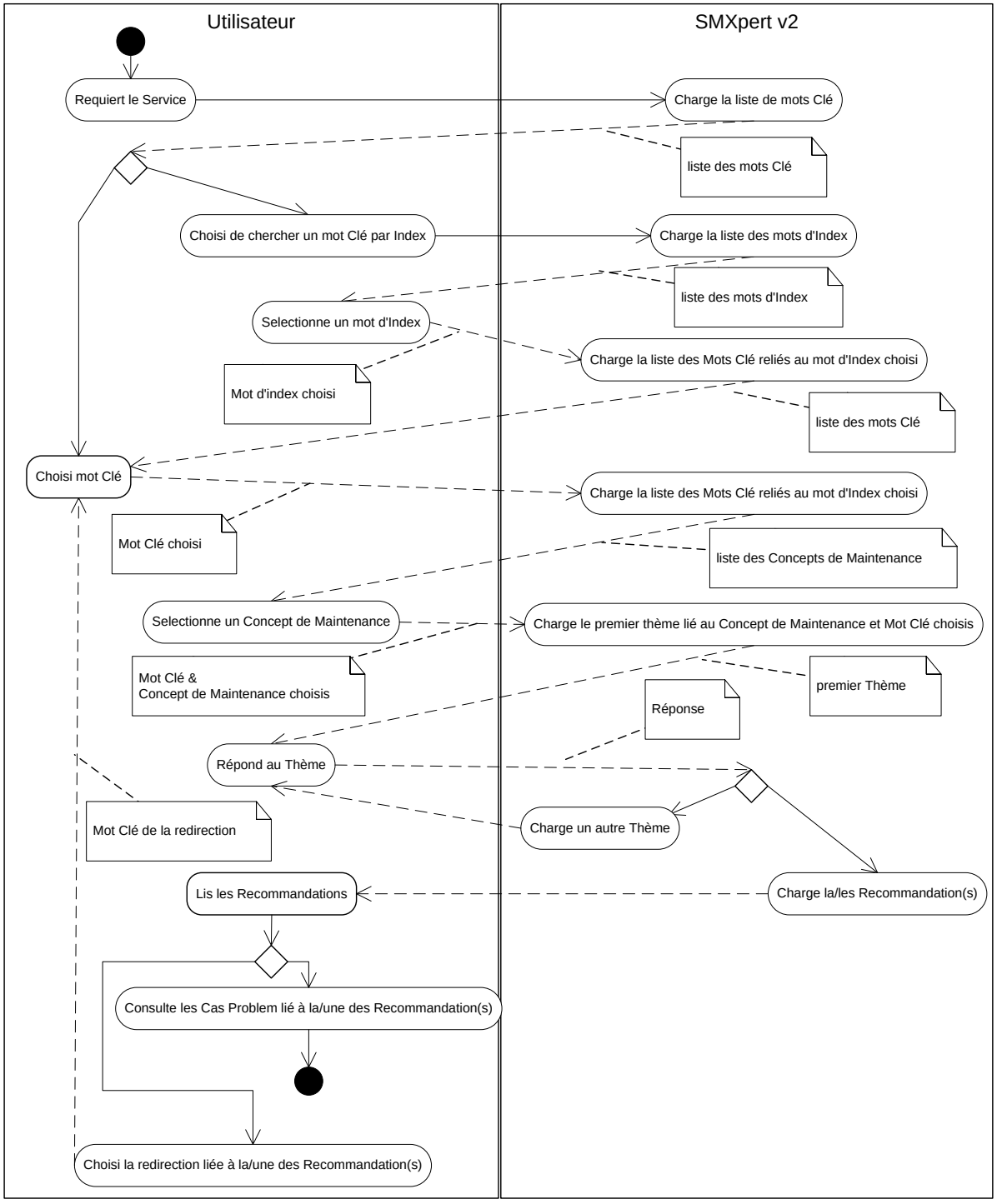
UML propose différentes manières de représenter un comportement. Il est intéressant de s'en servir pour étoffer cette analyse. Les cas d'utilisation offrant une vue encore trop abstraite de ce que fait le logiciel, nous allons expliquer comment le logiciel se comporte pour les différents cas d'utilisation.

Pour navigation dans la base des connaissances, la modélisation se fera par des diagrammes d'activités car ils mettent l'accent sur le séquencement/parallélisme, mettant en évidence les rôles de chacun dans une activité [14].

Pour ce qui est des activités d'évolution de la base de connaissances et la gestion des utilisateurs, le choix se dirige vers des diagrammes d'interaction car ils mettent l'accent sur les messages échangés entre objets/entité [14].

Tout le comportement du logiciel n'a pas été illustré à l'aide de diagrammes d'activité ou d'interaction. Seules les fonctionnalités implémentées sont reprises.

Navigation dans la base des connaissances

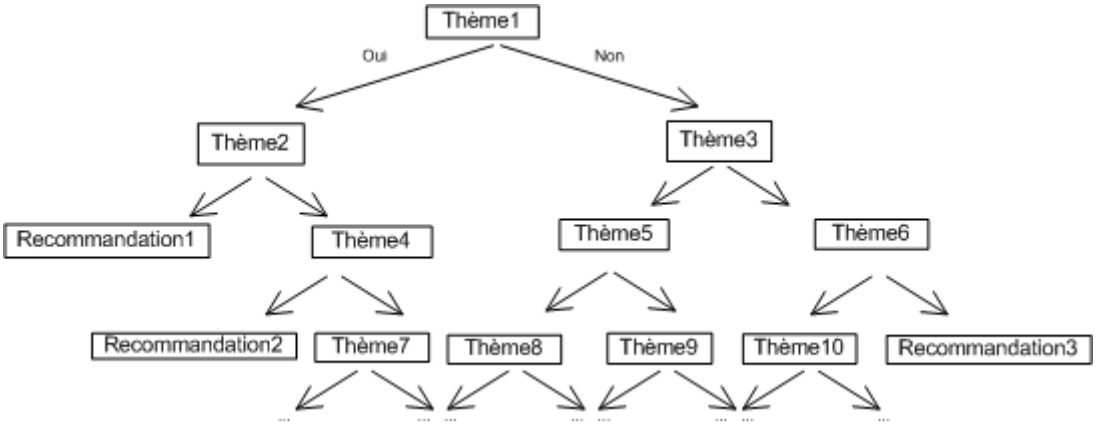


Le raisonnement par règle

Profitons de ce schéma pour illustrer le raisonnement du logiciel. Nous remarquons clairement que le raisonnement du logiciel *SM*^{Xpert} est à présent basé sur les règles. Le logiciel fonctionne par thème-réponse. Après le choix d'un mot-clé et d'un concept de maintenance, le

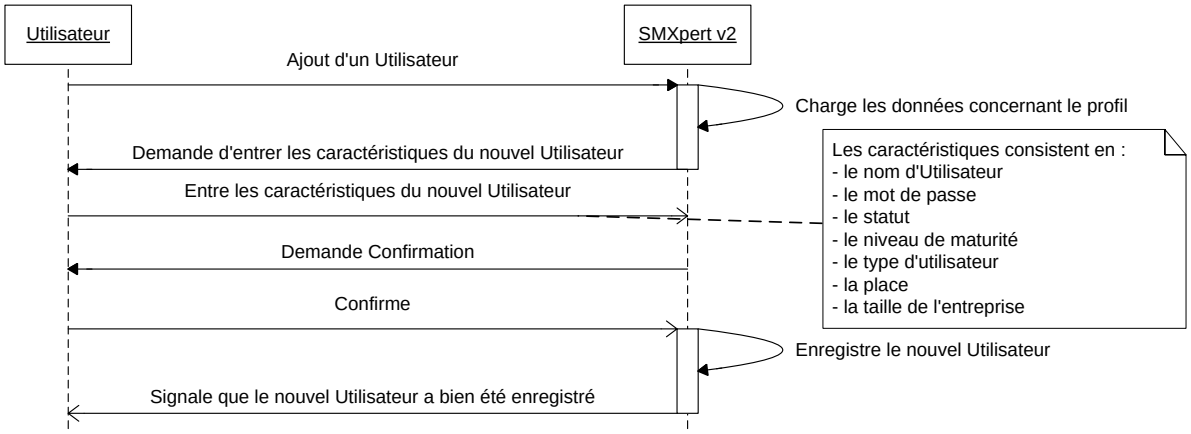
logiciel pose un thème et suivant la réponse de l'utilisateur, vérifie en mémoire s'il faut poser un autre thème ou fournir une recommandation.

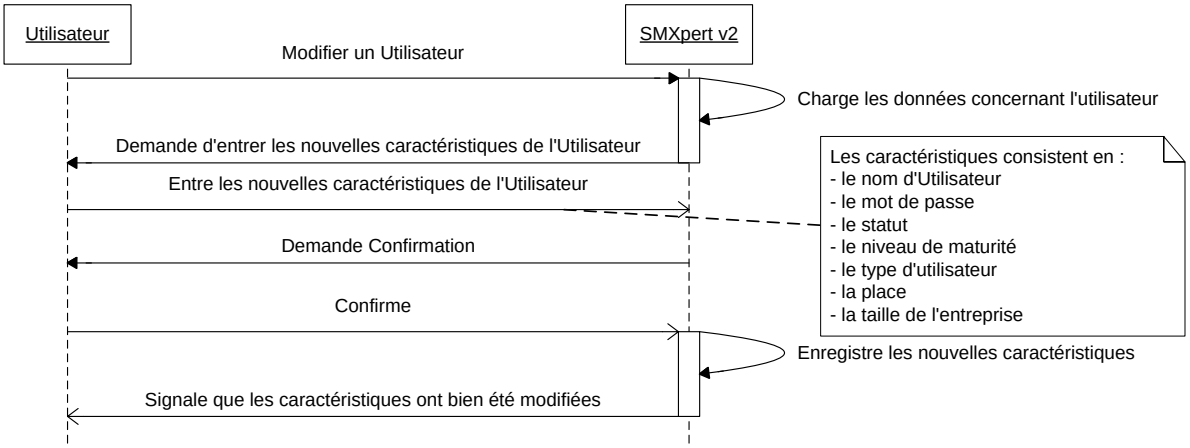
Concrètement le raisonnement s'apparente à un arbre de décision comme illustré sur le schéma suivant.

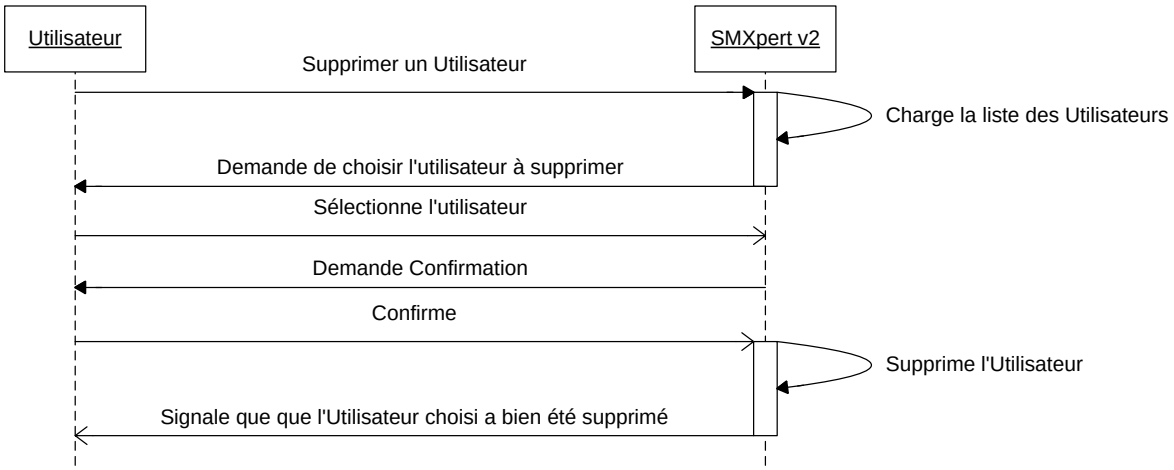


Gestion des utilisateurs

La gestion des utilisateurs peut être l'ajout d'un utilisateur de n'importe quel type, sa suppression et la modification de ses caractéristiques.





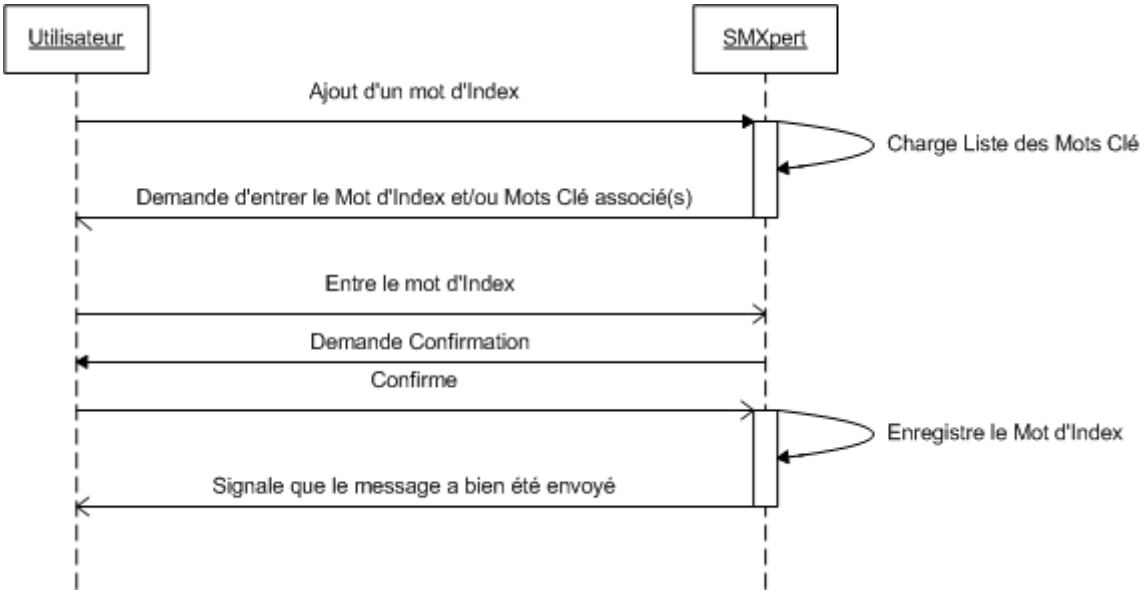


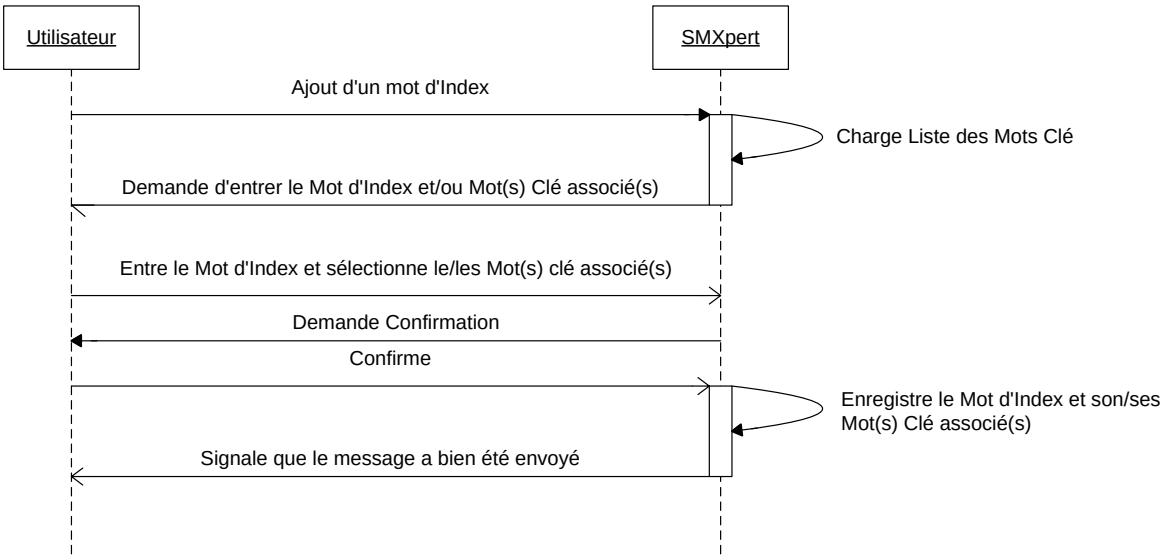
Changements de la base des connaissances

Les fonctionnalités permettant de changer la base des connaissances sont l'ajout d'un mot d'index, d'un mot-clé, d'un concept de maintenance, d'un thème, d'une recommandation et d'un cas problème.

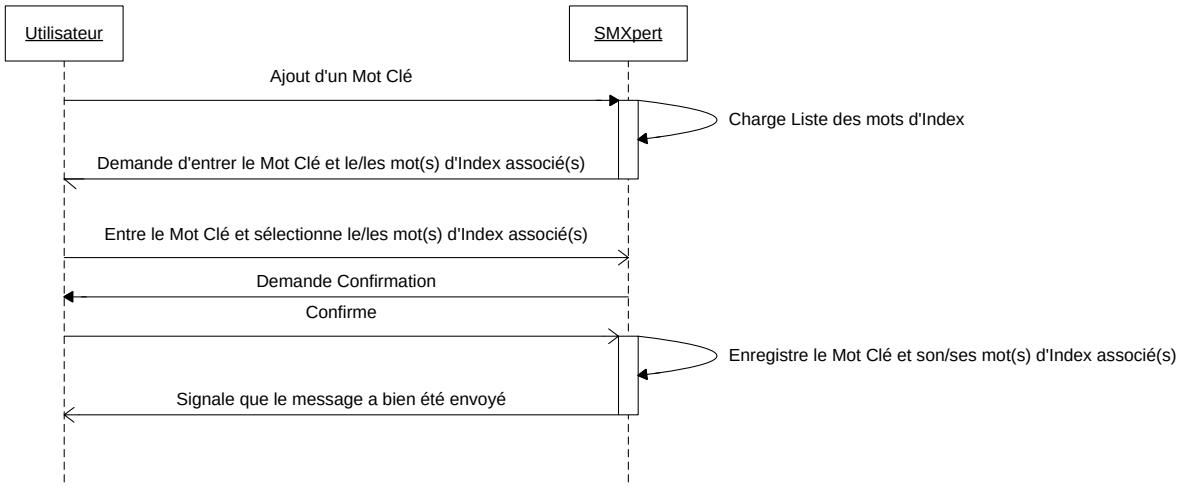
Ajout d'un mot d'index

Un mot d'index peut s'ajouter soit en le reliant à un mot-clé, soit en le laissant libre de tout lien avec un mot-clé. Cette seconde option n'est justifiée que par la nécessité de ne pas « bloquer » le logiciel ! En effet, sans cette option, il serait impossible de faire évoluer la base de connaissance si cette dernière était vide ! Le logiciel exigerait d'associer un mot d'index avec un mot-clé, mais il n'en existerait pas encore ; il serait donc impossible d'ajouter un mot d'index, et sans mot d'index, il est impossible d'ajouter un mot-clé. Sans mot-clé, pas d'ajout de concept de maintenance. Et ainsi de suite jusqu'à l'ajout d'un cas problème.

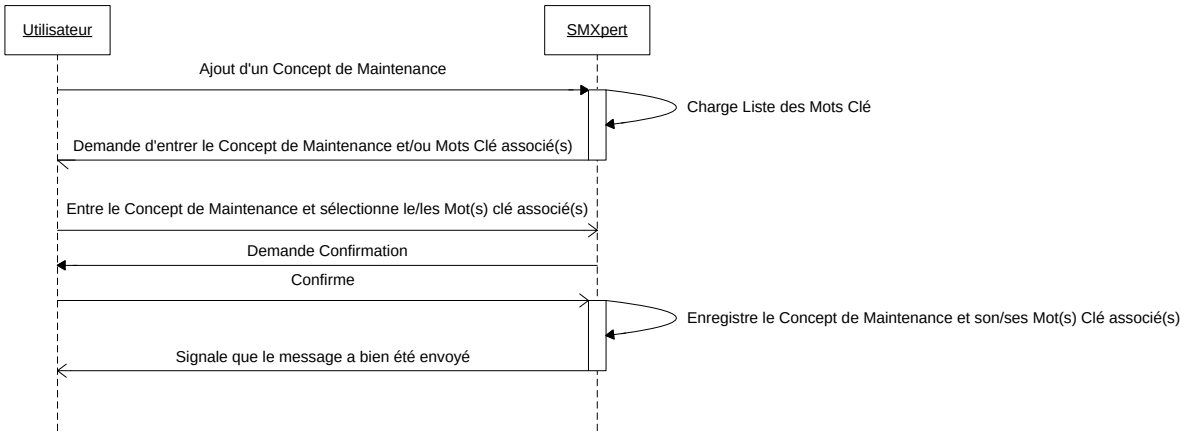

Utilisateur
SMXpert
Ajout d'un mot d'Index
Charge Liste des Mots Clé
Demande d'entrer le Mot d'Index et/ou Mots Clé associé(s)
Entre le mot d'Index
Demande Confirmation
Confirme
Enregistre le Mot d'Index
Signale que le message a bien été envoyé


Utilisateur
SMXpert
Ajout d'un mot d'Index
Charge Liste des Mots Clé
Demande d'entrer le Mot d'Index et/ou Mot(s) Clé associé(s)
Entre le Mot d'Index et sélectionne le/les Mot(s) clé associé(s)
Demande Confirmation
Confirme
Enregistre le Mot d'Index et son/ses Mot(s) Clé associé(s)
Signale que le message a bien été envoyé

Ajout d’un mot-clé



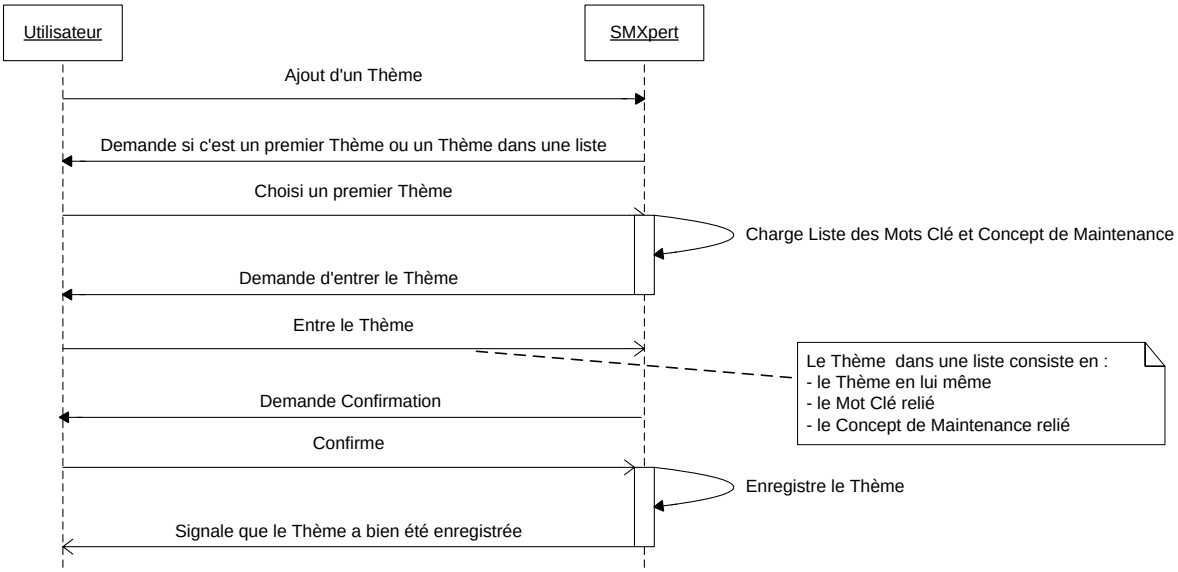
Ajout d’un concept de maintenance

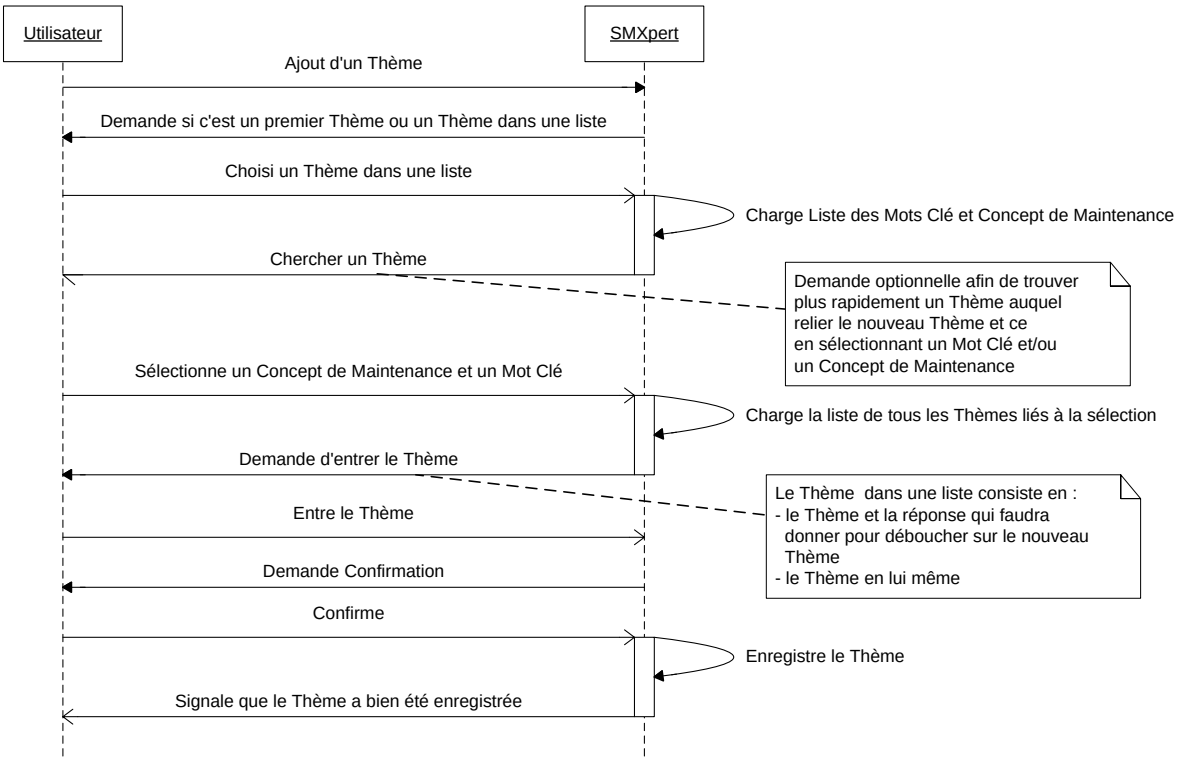


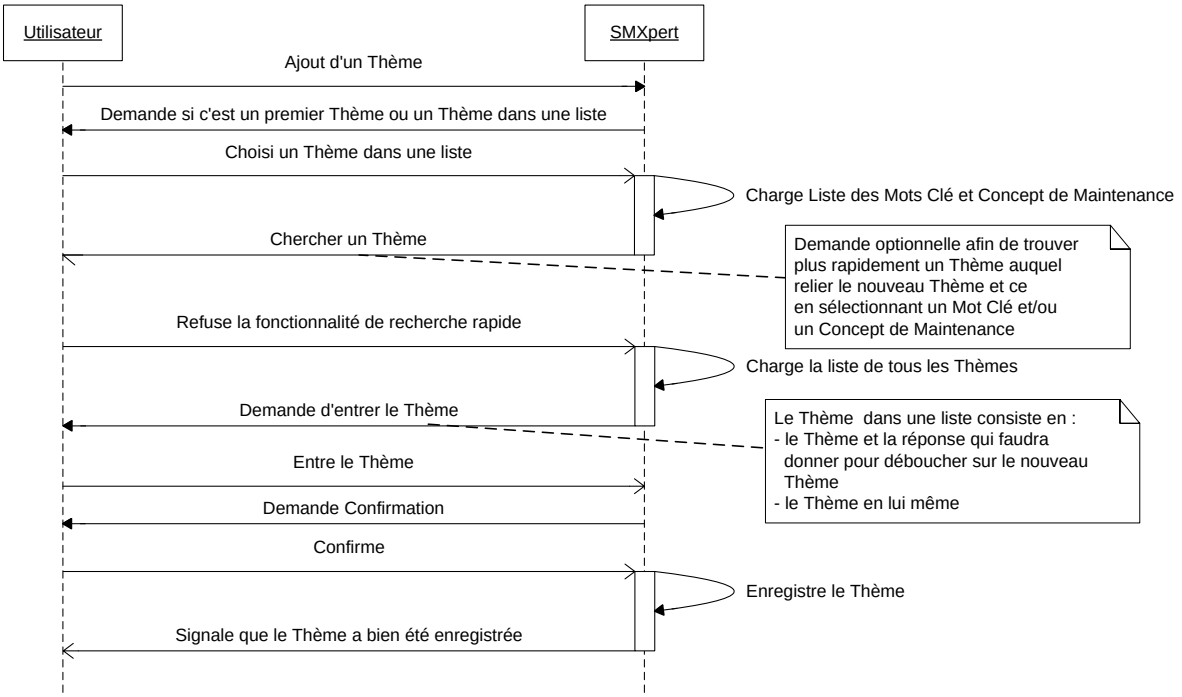
Ajout d’un Thème

Trois scénarios sont possibles pour l’ajout d’un thème, soit l’ajout d’un thème qui sera le premier d’une arborescence de thèmes, soit l’ajout d’un thème dans l’arborescence même. Il y a autant d’arborescences que de liens entre mots-clés et concepts de maintenance.

Pour la seconde option, le système offre la possibilité de trouver rapidement l’arborescence dans laquelle le thème doit être ajouté.

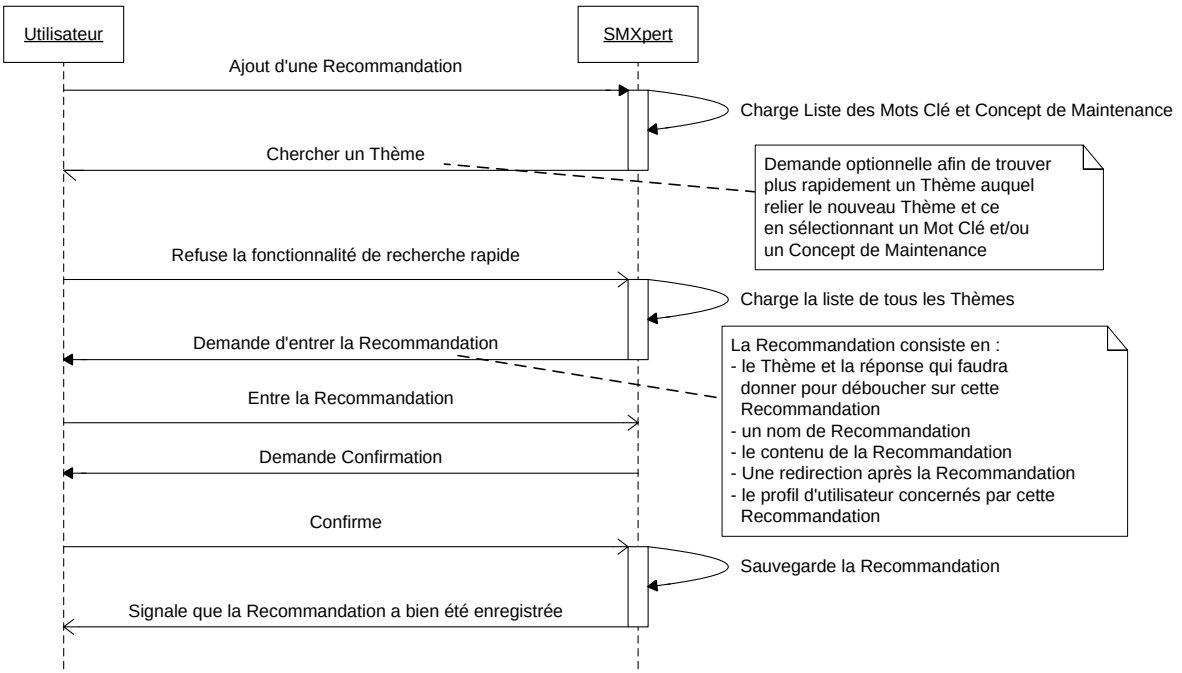

Utilisateur
SMXpert
Ajout d'un Thème
Demande si c'est un premier Thème ou un Thème dans une liste
Choisi un premier Thème
Charge Liste des Mots Clé et Concept de Maintenance
Demande d'entrer le Thème
Entre le Thème
Le Thème dans une liste consiste en :
- le Thème en lui même
- le Mot Clé relié
- le Concept de Maintenance relié
Demande Confirmation
Confirme
Enregistre le Thème
Signale que le Thème a bien été enregistrée

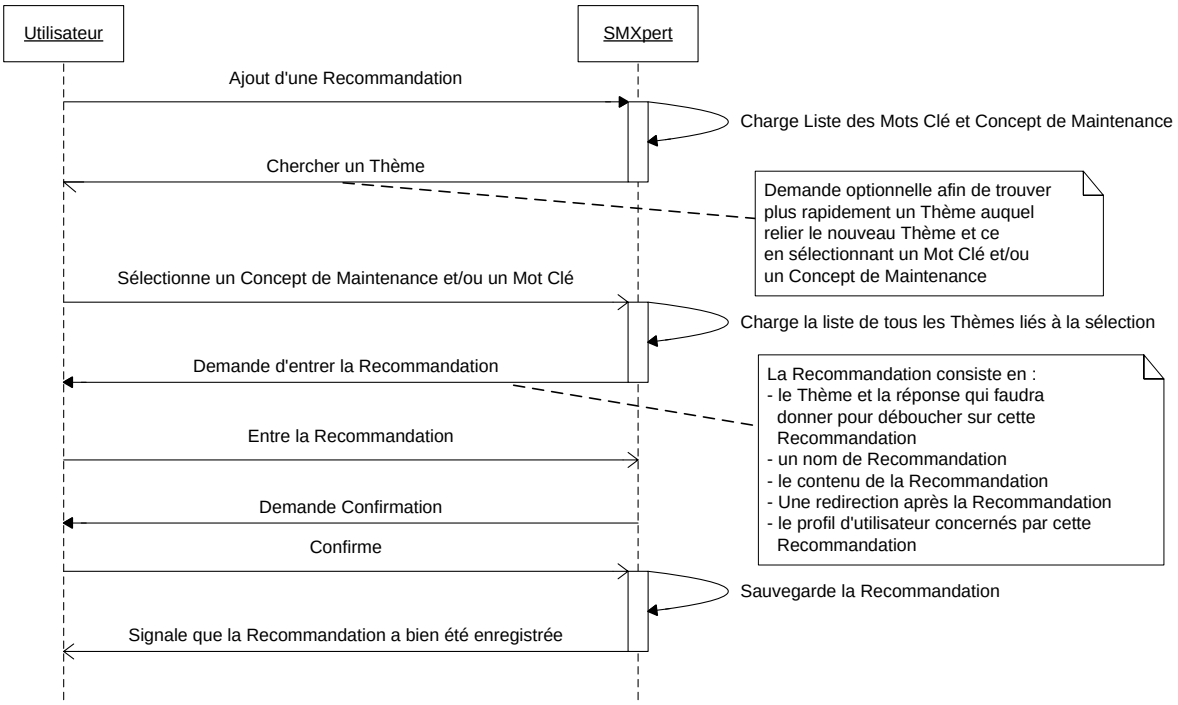

Utilisateur
SMXpert
Ajout d'un Thème
Demande si c'est un premier Thème ou un Thème dans une liste
Choisi un Thème dans une liste
Charge Liste des Mots Clé et Concept de Maintenance
Chercher un Thème
Demande optionnelle afin de trouver
plus rapidement un Thème auquel
relier le nouveau Thème et ce
en sélectionnant un Mot Clé et/ou
un Concept de Maintenance
Sélectionne un Concept de Maintenance et un Mot Clé
Charge la liste de tous les Thèmes liés à la sélection
Demande d'entrer le Thème
Le Thème dans une liste consiste en :
- le Thème et la réponse qui faudra
donner pour déboucher sur le nouveau
Thème
- le Thème en lui même
Entre le Thème
Demande Confirmation
Confirme
Enregistre le Thème
Signale que le Thème a bien été enregistrée



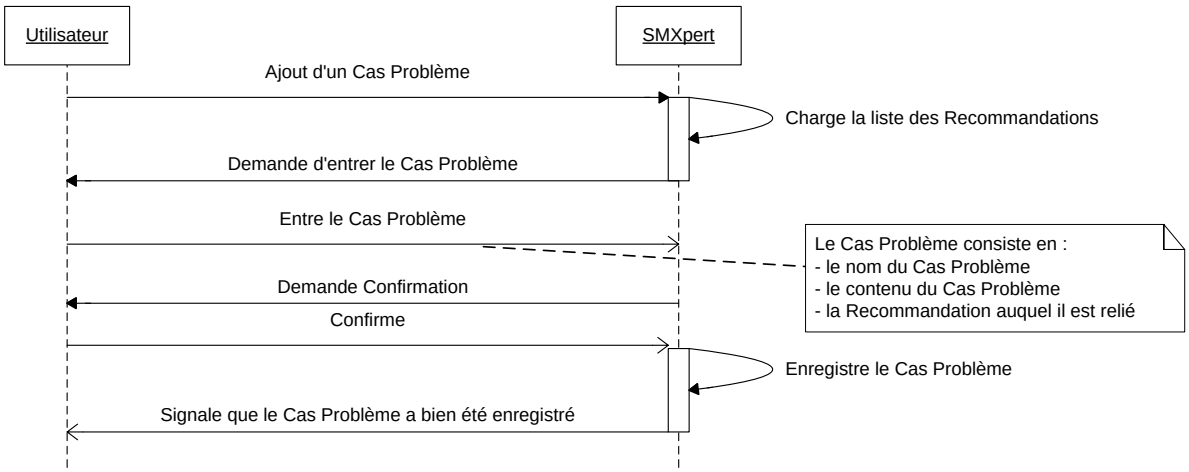
Ajout d'une recommandation

Deux scénarios sont possibles pour ajouter une recommandation. Une recommandation est associée à un thème se trouvant dans une arborescence ; le système offre optionnellement une possibilité de trouver plus rapidement l'arborescence du thème auquel l'utilisateur de type expert aimerait lier une recommandation.





Ajout d'un cas problème



6.5 Technologies utilisées

La plupart des technologies utilisées pour le développement du logiciel *SM*$^{Xpert\ v2}$ sont les mêmes technologies que pour l'ancienne version, c'est-à-dire les technologies Java, Java Servlets, l'outil Apache Tomcat qui sont expliqués dans [13] à la section 4.2.1, ainsi que le cadre de référence Struts permettant de créer des application Web expliqué [13] dans la section 4.5.1.

Il faut toutefois donner un aperçu de la technologie de mémorisation des données, le système de gestion de bases de données MySQL, car cette technologie n'est utilisée que pour la nouvelle conception.

MySQL

MySQL un serveur de bases de données relationnelles SQL très rapide, multi-thread, robuste et multi-utilisateurs. MySQL fonctionne sur beaucoup de plates-formes différentes comme Linux, Mac OS X et encore Windows, et les bases de données MySQL sont accessibles en utilisant le langage de programmation Java.

La version du serveur de bases de données relationnelles MySQL utilisée est MySQL 5.0.

« MySQL est le serveur de base de données le plus utilisé dans le monde. » [*Web07*] Surtout pour les applications Web [*Web08*].

Les principaux avantages de MySQL sont sa rapidité, sa robustesse et sa facilité d'utilisation. Ce sont ces avantages qui ont justifié le choix de MySQL comme système de gestion de base de données pour le logiciel SMXpert.

Toutefois MySQL possède un défaut, il est un peu court en fonctionnalités, mais ce défaut ne pose pas problème pour le logiciel *SM*^{Xpert}.

6.6 Comparaison

A présent que la nouvelle version a été présentée, il serait intéressant de la comparer avec l'ancienne version. Nous savons qu'un changement majeur a eu pour conséquence la prise de décision d'une ré-ingénierie, ce changement était la modification du déroulement de la navigation dans la page des utilisateurs de type « user ».

La nouvelle version du logiciel a été développée avec la plupart des mêmes outils que l'ancienne version et suit aussi, comme son prédécesseur, le *design pattern* « Model – View – Controller ».

Il est intéressant de comparer les schémas conceptuels, l'enchaînement des tâches, l'architecture logique abstraite et concrète des deux versions afin de bien cerner les résultats de la ré-ingénierie. Ces parties de l'analyse de *SM*$^{Xpert\ v2}$ n'ont pas été présentées dans une section propre, elles le sont ici afin de les comparer avec l'ancienne version. Cette comparaison sera faite dans le formalisme utilisé par S. Sandron et B. Vanderose pour l'ancienne version du logiciel.

Schémas conceptuels

Le schéma conceptuel de *SM*^{Xpert} se trouve au troisième chapitre à la Figure 18, section 2.4, celui de la nouvelle version est dans ce chapitre à la section 6.2.

Nous pouvons remarquer un point similaire, le lien entre index et mot-clé. Ensuite tout est différent. Il y a un lien entre mot-clé, concept de maintenance et thème, afin de mieux diriger un utilisateur dans sa navigation, ensuite les thèmes permettent de trouver quelle recommandation fournir à l'utilisateur. Il est donc dirigé d'une façon différente vers une recommandation, plus du tout par le choix d'un cas. De plus, les deux première étapes, celle

du choix d'un mot-clé et d'un concept de maintenance, ont plus d'importance que dans la seconde version. Par ailleurs cela constituait un problème conceptuel, le troisième problème conceptuel important relevé.

Nous voyons aussi que la notion de cas problème est totalement différente, moins important, il est aussi plus facile à présent d'insérer un cas problème, en l'associant uniquement à une recommandation.

Enchaînement des tâches

Voici le schéma de l'enchaînement des tâches de la nouvelle version du logiciel *SM*^{Xpert}. Le schéma de l'enchaînement des tâches de l'ancienne version se trouve au troisième chapitre, Figure 19, section 2.4.

Nous voyons que les deux premières tâches sont similaires ; cela vient de la seule similarité qu'il existe entre les deux schémas conceptuels.

Mais ensuite les déroulements respectifs deviennent différents, tout comme les schémas conceptuels diffèrent. Dans l'ancienne version, après les deux premières tâches, il fallait choisir un cas problème, ensuite répondre à une série de questions et le logiciel lui fournissait une recommandation. A présent, le logiciel pose un thème et, suivant la réponse du mainteneur, pose soit un autre thème, soit fournit une recommandation. Un autre problème conceptuel est aussi corrigé grâce à ce nouveau mode d'interaction : le fait que le logiciel pose une série de thèmes. Ensuite le mainteneur peut consulter facultativement un cas problème faisant office d'illustration par un exemple.

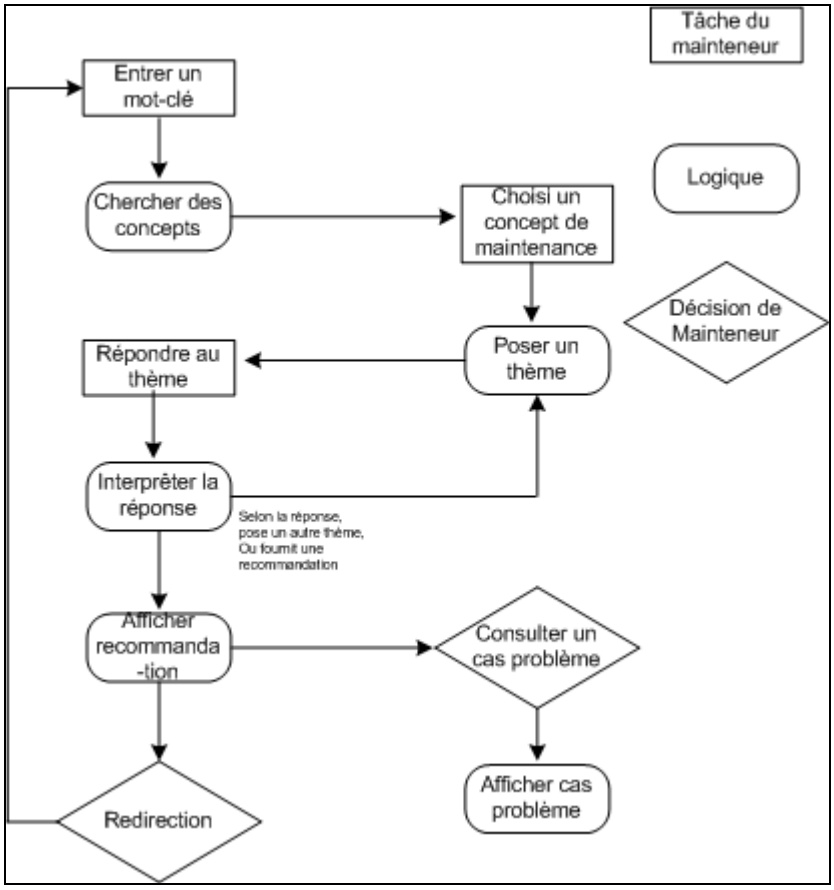


Figure 41 : Enchaînement des tâches de *SM*$^{Xpert\ v2}$

Architecture logique abstraite

Voici à présent l'architecture logique abstraite ainsi qu'un schéma des composants. L'architecture logique abstraire de l'ancienne version se trouve au troisième chapitre, Figure 20, section 2.4.

Nous pouvons constater que la seule différence entre les architectures logiques abstraites se trouve dans les composant du système à base de connaissances. Ceci est dû au fait d'avoir choisi le *design pattern* « Model – View – Controller ».

Nous pouvons constater sur les trois figures suivantes que par rapport à l'ancienne version les composants ont été faits suivant les utilisateurs et les fonctionnalités les concernant. La raison principale est de permettre facilement l'ajout ou la modification d'une fonctionnalité dans un composant.

Nous avions dans l'ancienne version, un composant « *index* » et un composant « *base de connaissance* », alors que l'index fait partie de la base des connaissances. Cette découpe remet donc aussi de l'ordre.

Enfin un composant gérant uniquement le système de gestion de base de données (*Data*) a été inséré ; il est utilisé par les trois autres composants et divisé suivant les besoins de chacun de ceux-ci.

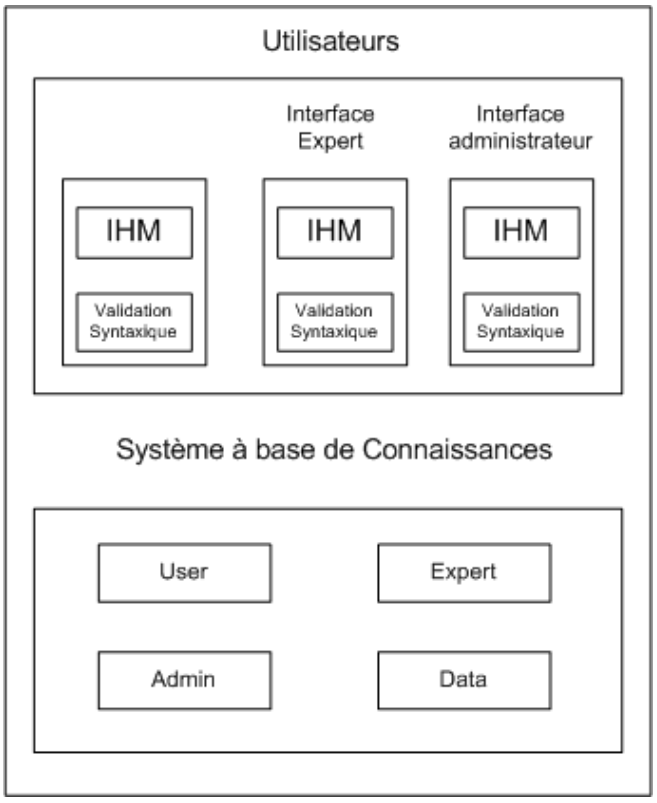


Figure 42 : Architecture logique abstraite de *SM*$^{Xpert\ v2}$

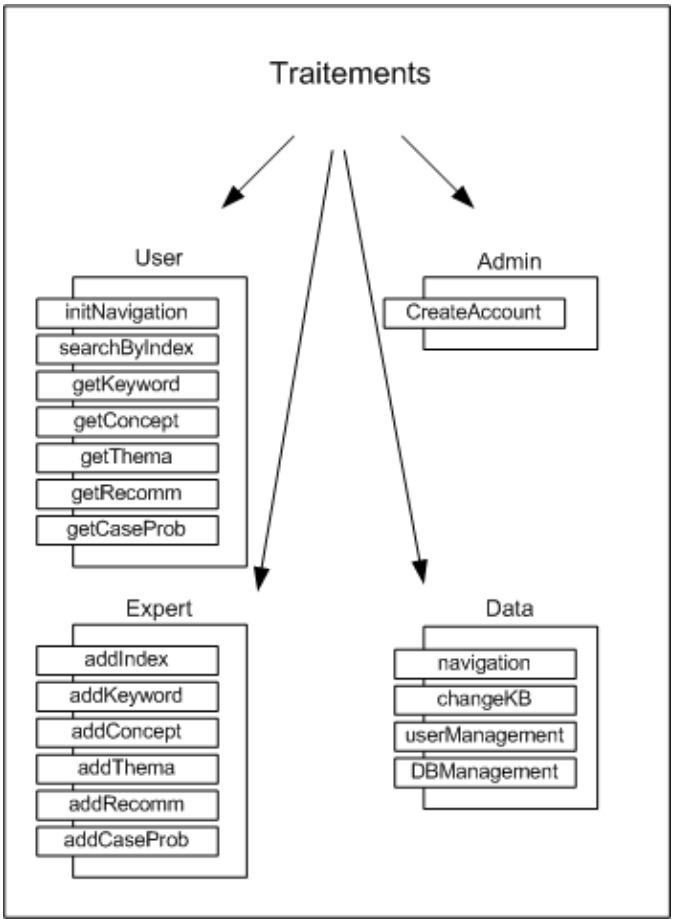


Figure 43 : schéma des composants

La Figure 43 nous montre également toutes les fonctionnalités implémentées dans cette version. Seules les fonctionnalités importantes, comme la navigation dans la base des connaissances et l'ajout dans la base des connaissances, ont pu être réalisées.

Architecture logique concrète

En ce qui concerne l'architecture logique concrète, il n'y a pas beaucoup de différence, c'est dû encore une fois au fait de l'utilisation du *design pattern* « Model – View – Controller ».

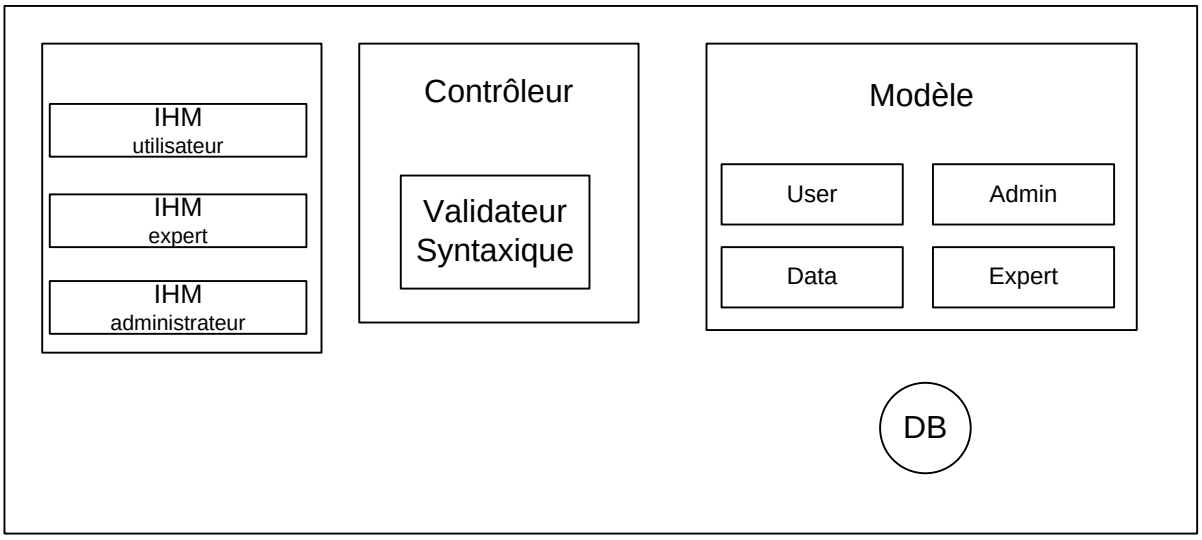


Figure 44 : Architecture logique concrète de *SM*$^{Xpert\ v2}$

Architecture physique

L'architecture physique est inchangée ; comme son ancienne version, le logiciel *SM*$^{Xpert\ v2}$ reste une application accessible par Internet, se trouvant sur un serveur Apache Tomcat et utilisant le cadre de référence Struts qui permet de créer des applications Web.

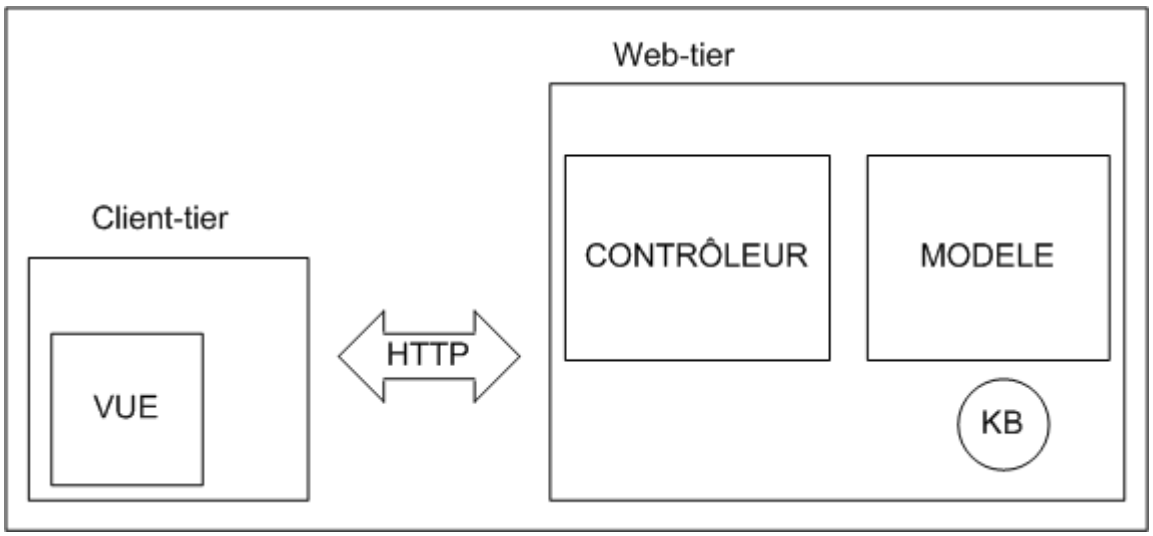


Figure 45 : Architecture physique de *SM*$^{Xpert\ v2}$

6.7 Conclusion

Nous avons à présent une nouvelle conception du logiciel *SM*$^{Xpert\ v2}$, avec une analyse des besoins en accord avec les experts en maintenance.

Le développement du logiciel a été réalisé avec les mêmes outils que sa précédente version, mise à part l'instauration d'un système de gestion de base de données.

Entre les deux versions, les différences essentielles sont celles du comportement du logiciel, et du schéma conceptuel. L'ancienne version était basée sur un raisonnement hybride, la nouvelle version est uniquement basée sur le raisonnement par cas. En ce qui

concerne les schémas conceptuels, seulement un point commun a été relevé, pour le reste tout est différent.

Nous voyons aussi qu'avec cette nouvelle conception, les problèmes conceptuels évoqués au chapitre 4 ont été corrigés.

Enfin les architectures concrète, abstraite et physique sont peu différentes entre les deux versions, car les architectures ne reflètent pas en réalité le comportement du logiciel ; de plus, les mêmes technologies ont été utilisées et le même *design pattern* a été adopté. Seuls les composants ont changé en étant redéfinis de manière plus orientée sur les trois types d'utilisateurs.

Conclusion et travaux futurs

SM^{Xpert} se veut un outil, d'aide à la décision, propre à la maintenance de logiciel, qui est à présent devenue une discipline à part entière malgré de nombreux liens avec le développement de logiciel. La maintenance a son propre cycle de vie, ses propres problèmes venant tant de l'extérieur que de l'intérieur, acteurs, outils de mesure et facteurs l'influençant.

C'est le fruit d'un travail de restructuration, effectué par messieurs S. Sandron et B. Vanderose, du logiciel COSMICXpert qui a donné naissance au logiciel *SM*^{Xpert}, un logiciel d'aide à la décision hybride combinant le raisonnement par cas et le raisonnement par règle.

Mais des plaintes concernant des difficultés et des incompréhensions émanant des utilisateurs ont abouti à une décision d'amélioration du logiciel. En effet, il était difficile l'utiliser en tant qu'outil d'aide à la décision mais principalement très difficile de faire évoluer sa base des connaissances.

Une analyse des faiblesses à partir des interfaces a été réalisée. Cette analyse a mis en avant des problèmes d'interface importants qu'il fallait résoudre pour faciliter l'utilisation du logiciel *SM*^{Xpert}. En outre cette analyse a aussi permis de découvrir des problèmes d'une autre nature, des problèmes d'ordre conceptuels. Enfin, de nouvelles propositions de changements sont venues s'ajouter. Pour améliorer le logiciel, il fallait aussi opérer tous ces changements.

Nous avons alors déterminé si les changements pouvaient se faire par une restructuration du logiciel ou bien une ré-ingénierie. Cette analyse a mis en évidence une modification importante : la modification du déroulement de la navigation dans la page des utilisateurs de type « user ». Cette modification change le fondement même du logiciel *SM*^{Xpert} et force un processus de ré-ingénierie qui paraissait trop audacieux au début de l'analyse du changement. C'est une ré-ingénierie faisant passer le logiciel *SM*^{Xpert} d'un raisonnement hybride vers un unique raisonnement par cas.

C'est ainsi qu'une nouvelle conception du logiciel a été élaborée : *SM*$^{Xpert\ v2}$. Cette nouvelle version diffère principalement de l'ancienne par un comportement volontairement différent et apporte une solution aux problèmes déterminés.

Les architectures concrète, abstraite et physique des deux versions ne présentent pas de différences importantes ; en réalité, elles ne reflètent pas le comportement du logiciel, le même *design pattern*, le « Model – View – Controller », a été adopté et le développement s'est réalisé à partir des mêmes technologies.

Cette nouvelle version répond aux exigences voulues, à la fois celles de départ en terme d'interfaces, d'erreurs et de problèmes d'ordre conceptuel mais également aux exigences visant une amélioration globale. Le logiciel est donc amélioré.

Notons cependant quelques imperfections. cette nouvelle version n'a été utilisée que par deux utilisateurs, dont un expert en maintenance, et il est difficile de cerner si le logiciel est bel et bien un système expert ou un système informationnel d'aide à la décision. Le logiciel possède bien une base de connaissance et un raisonnement par règle, mais les systèmes experts peuvent acquérir de nouvelles connaissances par l'ajout ou la mise en relation de nouvelles connaissances entre elles et avec les anciennes connaissances. Or il n'est possible ici que de faire évoluer la base des connaissances par ajout.

Bien que seulement deux personnes aient réellement utilisé la nouvelle version, elles ont validé son analyse. L'une de ces deux personnes étant un expert en maintenance, sa validation n'est pas un élément négligeable. Quelle personne serait mieux placée pour connaître les besoins d'un logiciel d'aide à la décision en maintenance qu'un expert ?
A présent *SM*$^{Xpert\ v2}$ est beaucoup plus facile à utiliser, et il est surtout possible de faire évoluer la base des connaissances par des procédures simples, beaucoup moins fastidieuses et plus intuitives.

Travaux Futurs

SM$^{Xpert\ v2}$ est loin d'être terminé ; déjà un nouveau projet est en cours, visant à améliorer le moteur d'inférence du logiciel qui, pour l'instant, se réduit à de la déduction par règle.

Il est nécessaire et important aussi de terminer l'implémentation de toutes les fonctionnalités déterminées à partir de l'analyse des besoins.

Il serait pertinent également de faire une analyse des interfaces du logiciel, à partir des critères relevés pour l'analyse de l'ancienne version. Cette analyse pourrait être réalisée par un panel d'utilisateurs de tout horizon. Ce qui validerait les interfaces et finaliserait ainsi le début et le but premier de ce projet.

Finalement, il serait intéressant d'ajouter deux fonctionnalités. Une première fonctionnalité à ajouter, dans l'évolution de la base des connaissances, serait de permettre l'ajout d'une arborescence entière de thèmes et de leurs recommandations, car présentement le logiciel ne permet que d'ajouter un seul thème et une seule recommandation à la fois. Une seconde, déjà évoquée, serait d'améliorer la page des utilisateurs de type « user », avec l'ajout d'une sorte de navigateur affichant l'endroit où se trouve l'utilisateur dans la base des connaissances.

Annexe 1 : Planification du projet

Ce projet a débuté par une familiarisation avec les interfaces du logiciel *SM*^{Xpert} afin d'avoir une idée des problèmes qui se posent aux utilisateurs, familiarisation qui a commencé par une première tâche consistant à entrer un cas problème dans la base de connaissance du logiciel *SM*^{Xpert}. La réussite de cette tâche a pris plusieurs essais. Les premiers essais consistaient à entrer des cas fictifs ; eut lieu ensuite l'entrée de cas de maintenance réels.

Une fois cette étape réussie, il est intéressant d'évaluer l'interface des utilisateurs de type « user ». Ces deux familiarisations ont permis de faire une définition du projet à réaliser comme indiqué au premier tableau (Définition du projet) ci-dessous.

Le projet est plus large que sa motivation initiale car il vise à améliorer l'entièreté du logiciel *SM*^{Xpert}, alors que la motivation était d'améliorer les interfaces.

Une fois la définition du projet établie, celui-ci a pu débuter. Comme il est indiqué sur le deuxième tableau ci-dessous (Planification du projet), le projet a débuté par une revue de littérature sur les interfaces permettant de faire une critique de l'interface actuelle du logiciel *SM*^{Xpert} et de définir une nouvelle solution. Après la définition de la nouvelle solution, il a fallut l'appliquer, ce qui est expliqué en détail sur le troisième tableau ci-dessous (Exécution).

Le quatrième tableau ci-dessous (Interprétation) explique comment l'évaluation du nouveau logiciel sera faite.

1.4.1 Définition du projet

Motivation	Objet	Objectif	Utilisateurs
Améliorer l'interface du système d'aide à la décision *SM*^{Xpert}.	Identifier, à partir des références pertinentes du domaine des interfaces utilisateurs de logiciel, les forces et les faiblesses de l'interface d'un logiciel d'aide à la décision (le logiciel *SM*^{Xpert}).	- Améliorer l'interface des utilisateurs de type « expert » actuelle. - Améliorer le modèle conceptuel du logiciel. - Améliorer la technique actuelle de gestion des données.	- Les personnes intéressées par le domaine des interfaces du logiciel. - Les experts de la maintenance. - Les utilisateurs de la maintenance - Les chercheurs en maintenance.

1.4.2 Planification du projet

Trois étapes ont été planifiées. Chaque étape se divise en 3.

Etapes	Intrants	Livrables
1 - Identifier dans les références pertinentes du domaine des interfaces utilisateurs de logiciel, l'information nécessaire à la conception d'interface de logiciel d'aide à la décision.	- Le modèle de maturité S^{3m}. Le logiciel *SM*^{Xpert} existant. - Différentes publications scientifiques pertinentes. - Les parties 10 à 12 de la norme ISO 9241.	- État de l'art sur les meilleures pratiques d'interfaces.
- Faire un sommaire de l'état de l'art, concernant les bonnes pratiques à utiliser		- Liste des principaux problèmes existants

pour concevoir des interfaces de logiciels d'aide à la décision.		de l'interface des utilisateurs de type « expert » du logiciel *SM*^{Xpert}.
	- Le cadre de référence pour l'amélioration de la qualité d'un logiciel d'aide à la décision (application à COSMICXpert et *SM*^{Xpert}.) développé par messieurs Stéphane Sandron et Benoit Vanderose.	- Proposition d'une nouvelle conception (uses cases, modèle de données, diagramme de classes, schéma entités-associations).
2 - Effectuer l'analyse des différents problèmes de l'interface actuelle du logiciel *SM*^{Xpert}, suivie d'une évaluation des impacts des changements du logiciel et de la technique actuelle de gestion de données.		- Étude de l'impact des changements sur le logiciel.
		- Choix des technologies recommandées pour satisfaire les exigences de la nouvelle version de *SM*^{Xpert}.
3 - Proposition d'une liste de modifications et d'ajouts pour résoudre les problèmes identifiés lors de l'étape 2.		

1.4.3 Exécution du projet

Préparation	Exécution	Analyse
Déploiement des technologies recommandées pour satisfaire les exigences de *SM*$^{Xpert\ v2}$.	Implémentation du système de gestion des utilisateurs de *SM*$^{Xpert\ v2}$.	Un logiciel où il est possible d'enregistrer son profil en tant qu'utilisateur de la maintenance à des niveaux différents, expert de la maintenance. Ainsi que s'enregistrer en tant qu'administrateur.
Création de la base de données ainsi que du patron d'accès permettant la communication avec cette dernière.	Implémentation des fonctionnalités permettant de naviguer dans la base de connaissance.	Un logiciel permettant l'aide à la décision, avec gestion de ses utilisateurs.
	Implémentation des fonctionnalité permettant d'accroître/changer la base de connaissance de *SM*$^{Xpert\ v2}$.	Un logiciel permettant l'aide à la décision, avec gestion de ses utilisateurs et pouvant évoluer au niveau de sa connaissance.
	Réalisation de l'interface du système de gestion des	

	utilisateurs. Réalisation de l'interface permettant d'accroître/changer la base de connaissance de *SM*$^{Xpert\ v2}$. Réalisation de l'interface de navigation dans la base de connaissance.	

Annexe 2 : Cas d'utilisation du logiciel *SM$^{Xpert\ v2}$*

Cette annexe présente une explication plus détaillée des schémas de cas d'utilisation du chapitre 6, c'est-à-dire que dans un but de clarification la majeure partie des cas d'utilisation seront expliqués.

Use case 1.0 **Créer un compte pour user**	
Pré : L'utilisateur de type « user » n'est pas encore enregistré dans le système. Le système est opérationnel.	
Déroulement	
Utilisateur de type « user »	Logiciel
1. Arrive sur la page d'accueil du logiciel en ligne.	
	2. Propose de se logger, de créer un nouveau compte ou de faire de la gestion de compte.
3. Demande pour créer un nouveau compte.	
	4. Demande des informations concernant l'utilisateur.
5. Entre ces informations.	
Post : l'utilisateur est enregistré dans le système et peut à présent se logger ou gérer son compte. Le système est opérationnel.	

Use case 2.0 (happy scénario) **Se logger (n'importe quel utilisateur)**	
Pré : L'utilisateur est déjà enregistré dans le système. Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Arrive sur la page d'accueil du logiciel en ligne.	
	2. Propose de se logger, de créer un nouveau compte ou de faire de la gestion de compte.
3. Demande de se logger.	
	4. Demande son nom d'utilisateur et son mot de passe.
5. Entre son nom d'utilisateur et son mot de passe.	
	6. Détecte le statut suivant le nom d'utilisateur et dirige l'utilisateur sur la page le concernant.
Post : l'utilisateur est sur le système. Le système est opérationnel.	

Use case 2.1 (mauvais loggin ou mot de passe) **Se logger cas alternatif (n'importe quel utilisateur)**	
Pré : L'utilisateur est déjà enregistré dans le système. Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Arrive sur la page d'accueil du logiciel en ligne.	
	2. Propose de se logger, de créer un nouveau compte ou de faire de la gestion de compte.
3. Demande de se logger.	
	4. Demande son nom d'utilisateur et son mot de passe.

5. Entre son nom d'utilisateur et son mot de passe.	
	6. Détecte que le loggin n'existe pas ou que le mot de passe est incorrect.
	7. Signale à l'utilisateur que son loggin ou son mot de passe est incorrect.
	8. Demande de taper à nouveau son loggin et mot de passe.
Retour Use case1.1 point 4	

Use case 3.0 **Gérer son compte (n'importe quel utilisateur)**	
Pré : L'utilisateur est déjà enregistré dans le système. Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Arrive sur la page d'accueil du logiciel en ligne.	
	2. Propose de se logger, de créer un nouveau compte ou de faire de la gestion de compte.
3. Choisit de faire de la gestion de compte.	
	4. Demande son nom d'utilisateur et son mot de passe.
5. Entre son nom d'utilisateur et son mot de passe.	
	6. Affiche les informations concernant l'utilisateur.
7. Opère les modifications voulues et valide son action.	
	8. Demande confirmation.
9. Confirme.	
Post : Le profil de l'utilisateur a changé. Le système est opérationnel.	

Use case 4.0 **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user.	
Déroulement	
Utilisateur	Logiciel
1. Est sur la page des utilisateurs de type user.	
	2. Propose d'entrer un mot-clé ou de chercher un mot-clé par index.
3. Suivant le choix de l'utilisateur aller aux uses cases 4.0.x.	
	4. Reçoit le mot-clé et propose une série de concept de maintenance.
5. Choisit un concept de maintenance.	
	6. Propose une première question suivant le mot-clé et le concept ainsi que le profil de l'utilisateur.
7. Répond à la question.	
Suivant la réponse de l'utilisateur aller aux uses cases 4.x	
Post : l'utilisateur a une recommandation.	

Use case 4.0.1 (choisit un mot-clé) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user et le système lui propose d'entrer un mot-clé ou de chercher un mot-clé par index.	
Déroulement	
Utilisateur	Logiciel
1. Entre son mot-clé.	
Retour au use case 4.0 étape 4	

Use case 4.0.2 (cherche par l'index) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user et le système lui propose d'entrer un mot-clé ou de chercher un mot-clé par index.	
Déroulement	
Utilisateur	Logiciel
1. Demande de trouver un mot-clé par l'index.	
	2. Propose de rentrer un mot.
3. Rentre un mot (il ne sera pas possible qu'il rentre un mot n'existant pas dans l'index).	
	4. Propose les mots-clés en rapport avec le mot d'index.
5. Choisit le mot-clé lui convenant.	
Retour au use case 4.0 étape 4	

Use case 4.1 (propose une deuxième question) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user. L'utilisateur a répondu à une question du logiciel.	
Déroulement	
Utilisateur	Logiciel
	1. Propose une autre question.
Retour use cas 4.0 étape 7	

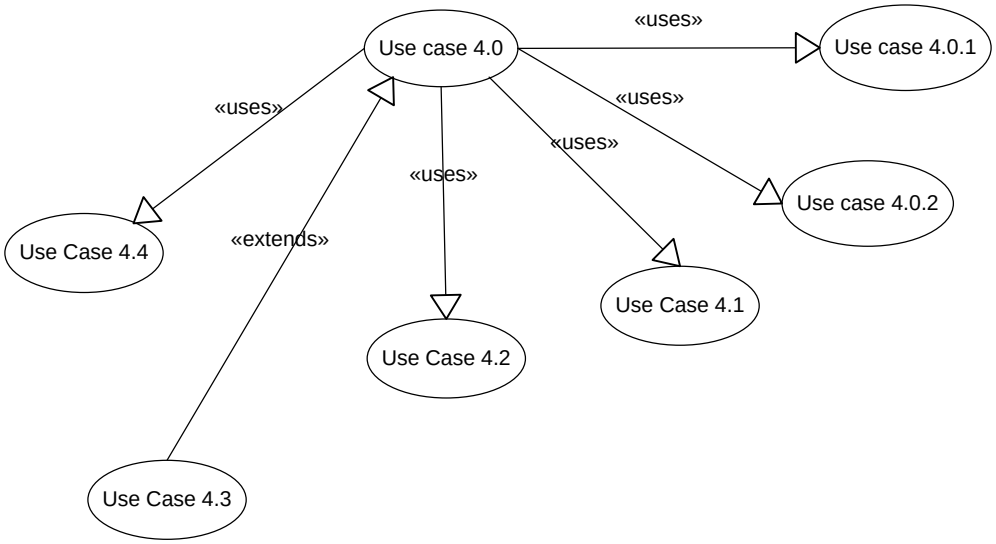
Use case 4.2 (reçoit une recommandation) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user. L'utilisateur a répondu à une question du logiciel.	
Déroulement	
Utilisateur	Logiciel
	1. Propose une recommandation.
Retour use cas 4.0 à la post condition	

Use case 4.3 (reçoit une recommandation et un cas problème) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type user. L'utilisateur a répondu à une question du logiciel.	
Déroulement	
Utilisateur	Logiciel
	1. Propose une recommandation et un cas problème en rapport avec la recommandation fournie.

Post : l'utilisateur a une recommandation ainsi qu'un cas problème explicatif en rapport avec la recommandation.

Use case 4.4 (prend une redirection) **Navigation dans la base des connaissances (n'importe quel utilisateur)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page de navigation dans la connaissance (à mieux exprimer).	
Déroulement	
Utilisateur	Logiciel
	1. Suivant la réponse de l'utilisateur, propose une recommandation ainsi qu'une redirection.
Si l'utilisateur prend la redirection, alors retour au use case 4.0 étape 4 ; si pas, alors retour au use case 4.0 post condition.	

Pour comprendre correctement le cas d'utilisation « Navigation dans la base des connaissances », il est intéressant de fournir un schéma montrant comment s'agencent tous les sous-cas d'utilisation.



Use case 5.0 **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour modifier la base des connaissances.	
	2. Le logiciel demande si l'expert veut ajouter, modifier ou supprimer un thème, un cas problème, une recommandation, un concept de maintenance, un mot clé ou un mot d'index.
Suivant la réponse de l'expert aller à use case 5.x	

Use case 5.1 (ajout mot d'index) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.

Déroulement	
Utilisateur	Logiciel
1. Demande pour ajouter un mot d'index.	
	2. Le logiciel demande d'entrer le mot d'index et de le lier à un ou plusieurs mots d'index.
3. Entre le mot d'index et, s'il le désire, sélectionne le ou les mots-clés reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a inséré un nouveau mot d'index, la base de connaissance a été modifiée et le système est opérationnel.	

Use case 5.2 (ajout mot-clé) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour ajouter un mot-clé.	
	2. Le logiciel demande d'entrer le mot-clé et de le lier à un ou plusieurs mots d'index.
3. Entre le mot-clé et sélectionne le ou les mots d'index reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a inséré un nouveau mot-clé, la base de connaissance a été modifiée et le système est opérationnel.	

Use case 5.3 (ajout concept de maintenance) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour ajouter un concept de maintenance.	
	2. Le logiciel demande d'entrer le concept de maintenance et de le lier à un ou plusieurs mots-clés.
3. Entre le concept de maintenance et sélectionne le ou les mots-clés reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a inséré un nouveau concept de maintenance, la base de connaissance a été modifiée et le système est opérationnel.	

Use case 5.4 (ajouter un thème) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour ajouter un thème.	

	2. Le logiciel lui demande à quel concept de maintenance et mot-clé le thème doit être associé, ainsi qu'un ou des mots-clés.
3. Coche le concept de maintenance et le mot-clé associé.	
	4. Enregistre les informations, puis demande d'encoder le thème.
5. Encode le thème.	
	6. Enregistre l'information et ensuite demande la place du thème dans l'arborescence associée au concept de maintenance et mot-clé précédemment choisis.
7. Place le thème dans l'arborescence liée au concept de maintenance et mot-clé précédemment choisis.	
	8. Demande les informations de déroulement lors de la réponse à ce nouveau thème : ce qui se passe lors d'une réponse positive et ce que se passe lors d'une réponse négative.
9. Opère les changements et liens nécessaires. Soit : indique si une réponse positive ou négative débouche sur un autre thème ou sur une recommandation.	
	10. Enregistre les informations.
Post : L'utilisateur a opéré ses changements, la base de connaissance a été modifiée et le système est opérationnel.	

Use case 5.5 (ajouter une recommandation) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. L'expert demande pour associer une recommandation à un thème.	
	2. Propose facultativement une méthode de recherche de l'arborescence (suivant son concept de maintenance et/ou mot-clé) du thème auquel la recommandation sera ajoutée.
Suivant le choix de recherche de l'utilisateur, aller aux uses cases 5.5.x	

Use case 5.5.1 (choisit la recherche de l'arborescence) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé d'ajouter une recommandation.	
Déroulement	
Utilisateur	Logiciel
1. Choisit le concept de maintenance et/ou mots-clés auxquels sont associés l'arborescence du thème dont l'utilisateur veut associer une recommandation.	
	2. Propose la liste des arborescences associées au concept de maintenance et/ou mots-clés.
3. Choisit le thème.	
	4. Propose d'entrer la recommandation.
5. Entre la recommandation, ainsi que le profil d'utilisateur auquel il est convenu de fournir cette information. Ensuite valide.	
	6. Le logiciel demande confirmation.
7. L'utilisateur confirme.	

Post : L'utilisateur a opéré ses changements, la recommandation voulue a été insérée et le système est opérationnel.

Use case 5.5.2 (choisit dans la liste complète) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé d'ajouter une recommandation.	
Déroulement	
Utilisateur	Logiciel
1. Choisit le thème, dans la liste complète de tous les thèmes, auquel sera ajoutée la recommandation.	
	2. Propose d'entrer la recommandation.
3. Entre la recommandation, ainsi que le profil d'utilisateur auquel il est convenu de fournir cette information. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a opéré ses changements, la recommandation voulue a été insérée et le système est opérationnel.	

Use case 5.6 (ajout cas problème) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour ajouter un cas problème.	
	2. Le logiciel demande d'entrer le cas problème et de le lier à une recommandation.
3. Entre le cas problème et sélectionne la recommandation reliée. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a inséré un nouveau mot d'index, la base de connaissance a été modifiée et le système est opérationnel.	

Use case 5.7 (Changer mot d'index) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour changer un mot d'index.	
	2. Le logiciel demande de choisir le mot d'index à modifier.
3. Choisit le mot d'index puis entre le nouveau mot d'index et, s'il le désire, sélectionne le ou les nouveaux mots-clé- reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a modifié le mot d'index voulu et le système est opérationnel.	

Use case 5.8 (Changer mot-clé) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**

Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour changer un mot-clé.	
	2. Le logiciel demande de choisir le mot-clé à modifier.
3. Choisi le mot-clé à modifier et puis entre le nouveau mot-clé et sélectionne le ou les nouveaux mots d'index reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a modifié le mot-clé voulu et le système est opérationnel.	

Use case 5.9 (Changer concept de maintenance) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour changer un concept de maintenance.	
	2. Le logiciel demande de choisir le concept de maintenance à modifier.
3. Choisit le concept de maintenance à modifier puis entre le nouveau concept de maintenance et sélectionne le ou les nouveaux mots d'index reliés. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a modifié le mot concept de maintenance voulu et le système est opérationnel.	

Use case5.10 (changer un thème) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande de modifier un thème.	
	2. Propose plusieurs méthodes de recherche du thème à modifier. Soit par la liste des thèmes, soit par l'arborescence dans laquelle se trouve le thème voulu. La recherche de l'arborescence se fait par le choix d'un concept de maintenance et/ou mots-clés.
Suivant le choix de recherche de l'utilisateur aller aux uses cases 5.10.x	

Use case 5.10.1 (choisit liste des thèmes) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé de modifier un thème.	
Déroulement	
Utilisateur	Logiciel

1. Choisit le thème à modifier dans la liste des thèmes.	
Aller à use case 5.10 suite 1	

Use case 5.10.2 (choisit le concept de maintenance et/ou mots-clés) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a demandé de modifier un thème.	
Déroulement	
Utilisateur	Logiciel
1. Choisit le concept de maintenance et/ou mot-clé auxquels le thème est associé.	
	2. Propose l'arborescence de thèmes associée au concept de maintenance et/ou mots-clés choisis par l'utilisateur.
3. Choisit le thème à modifier.	
Aller à use case 5.10 suite 1	

Use case 5.10 suite 1 (modifier un thème ; suite) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a choisi un thème à modifier dans le processus de modification de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
	1. Demande quelle est la modification voulue pour le thème choisi : - modifier le texte du thème ; - modifier le déroulement d'une réponse à un thème ; - supprimer une recommandation associée à un thème et actualiser le déroulement des réponses à ce thème.
Suivant le choix de modification de l'utilisateur aller aux uses cases 5.10 suite 1.x	

Use case 5.10 suite 1.1 (modifier texte d'un thème ; suite) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a choisi de modifier le texte d'un thème dans le processus de modification de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour modifier le texte d'un thème.	
	2. Demande d'entrer la modification.
3. Modifie le thème.	
	4. Sauvegarde les modifications.
Post : L'utilisateur a opéré ses changements, le texte du thème voulu a été modifié, le système est opérationnel.	

Use case 5.10 suite 1.2 (modifier déroulement) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a choisi de modifier le déroulement d'un thème dans le processus de modification de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour modifier le déroulement d'une réponse à un thème.	

	2. Demande les informations de déroulement : ce qui se passe lors d'une réponse positive et ce que se passe lors d'une réponse négative.
3. Opère les changements et liens nécessaires. C'est-à-dire : indique si une réponse positive ou négative débouche sur un autre thème ou sur une recommandation.	
	4. Enregistre les informations.
Post : L'utilisateur a opéré ses changements, le déroulement de la question voulue a été modifié, le système est opérationnel.	

Use case 5.10 suite 1.3 (supprimer une recommandation) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin< ; »)**	
Pré : Le système est opérationnel et l'utilisateur a choisi de supprimer une recommandation associée à une question dans le processus de modification de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour supprimer une recommandation.	
	2. Supprime la recommandation et demande d'actualiser de déroulement : ce qui se passe lors d'une réponse positive et ce que se passe lors d'une réponse négative.
3. Opère les changements et liens nécessaires. C'est-à-dire : indique si une réponse positive ou négative débouche sur un autre thème ou sur une recommandation.	
	4. Enregistre les informations.
Post : L'utilisateur a opéré ses changements, la recommandation voulue a été supprimée et le déroulement a été actualisé, le système est opérationnel.	

Use case 5.11 (changer une recommandation) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur a demandé de faire des modifications de la base des connaissances.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour modifier une recommandation.	
	2. Propose plusieurs possibilités pour rechercher la recommandation à modifier. Soit par la liste de toutes les recommandations. Soit en choisissant le thème auquel est associée la recommandation à modifier.
Suivant le choix de recherche de l'utilisateur aller aux uses cases 5.11.x	

Use case 5.11.1 (choisi dans la liste) **Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)**	
Pré : Le système est opérationnel et l'utilisateur est sur la page principale des utilisateurs de type expert. L'utilisateur a demandé de modifier une recommandation.	
Déroulement	
Utilisateur	Logiciel
1. Choisit une recommandation dans la liste.	
	2. Propose d'entrer la nouvelle recommandation.

3. Entre la nouvelle recommandation, ainsi que le profil d'utilisateur auquel il est convenu de fournir cette information. Ensuite valide.	
	4. Enregistre la nouvelle recommandation.
Post : L'utilisateur a opéré ses changements, la recommandation voulue a été modifiée et le système est opérationnel.	

Use case 5.11.2 (choisi le thème auquel est associée la recommandation) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel et l'utilisateur a demandé de modifier une recommandation.	
Déroulement	
Utilisateur	Logiciel
1. Choisit le thème auquel est associée la recommandation à modifier.	
	2. Propose la liste des recommandations associées au thème choisi par l'utilisateur.
3. Choisit la recommandation à modifier.	
	4. Propose d'entrer la nouvelle recommandation.
5. Entre la nouvelle recommandation, ainsi que le profil d'utilisateur auquel il est convenu de fournir cette information. Ensuite valide.	
	6. Enregistre la nouvelle recommandation.
Post : L'utilisateur a opéré ses changements, la recommandation voulue a été modifiée et le système est opérationnel.	

Use case 5.12 (Changer un cas problème) Evolution de la base de connaissance. (Utilisateur de type « expert » et « admin. »)	
Pré : Le système est opérationnel.	
Déroulement	
Utilisateur	Logiciel
1. Demande pour changer un cas problème.	
	2. Le logiciel demande de choisir le cas problème à modifier parmi la liste des cas problèmes.
3. Choisit le cas problème puis entre les nouvelles informations. Ensuite valide.	
	4. Le logiciel demande confirmation.
5. L'utilisateur confirme.	
Post : L'utilisateur a modifié le cas problème voulu et le système est opérationnel.	

Pour comprendre correctement le cas d'utilisation « Evolution de la base de connaissance », il est intéressant de fournir un schéma montrant comment s'agencent tous les sous-cas d'utilisation.


Use case 5.1
Use case 5.2
Use case 5.8
Use case 5.3
Use case 5.9
Use case 5.9
Use case 5.4
Use case 5.8
Use case 5.0
Use case 5.5
Use case 5.5.1
Use case 5.7
Use case 5.10
Use case 5.5.2
Use case 5.6
Use case 5.11
Use case 5.10.1
Use case 5.10.2
Use case 5.10 suite 1
Use case 5.10 suite 1.1
Use case 5.10 suite 1.2
Use case 5.10 suite 1.3
«extends»

Références

[1] **J-M Jacquet**, « Technique d'Intelligence Artificielle » - *Namur : Facultés universitaires Notre-Dame de la Paix, Année académique 2004-2005.*
[2] **A Abran et J-M Desharnais**, « Applying a Functional Measurement Method : Cognitive Issues » - *Montréal, Quebec, Canada : International Workshop on Software Measurement (IWSM'01), 28-29 août 2001.*
[3] **Li D. Xu**, « Case-Based Reasonning : A major paradigm of artificial intelligence » - *IEEE Potentials, 1994.*
[4] **L. An, J. Yan et L. Tong,** « An Intergrated Rule-Based and CaseBased Reasoning System for Customer Service Management » - *IEEE International Conferene an e-Business Enguneering (ICEBE'05), 2005*.
[5] **E. J. Chikofsky et J. H. Cross**, « Reverse Engineering and Design Recovery: a Taxonomy » - *IEEE, janvier 1990*.
[6] **S. Perera et I. Watson**, « An integrated approach for Case-Based design and estimating » - *London : IEE, 1995*.
[7] **J-M Desharnais, A Abran, A Mayers, L. Buglione et V. Bevo**, « Knowledge Modeling for the Design of a KBS in the functional Size Measurement Domain ».
[8] **M. Noirhomme – Fraiture**, « Human Computer Interaction » - *Namur : Facultés universitaires Notre-Dame de la Paix, Année académique 2003-2004*.
[9] **Association Française de Normalisation**, « exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation – Partie 10 » - *Paris La Défense Cedex, 1996.*
[10] **Association Française de Normalisation**, « exigences ergonomiques pour travail de bureau avec terminaux à écrans de visualisation – Partie 12 » - *Paris La Défense Cedex, 1999.*
[11] **A. April et A. Abran**, « Modèle d'Evaluation de la Capacité à Maintenir le Logiciel (MÉC^{ML}) » - *Université de Québec : Ecole de Technologie Supérieure, Montréal, Québec, Canada, 2005.*
[12] **B. A. Kitchenham et Al.**, « Towards an Ontology of Software Maintenance » - *Journal of Software Maintenance : Research and Practise, 1999.*
[13] **S. Sandron et B. Vanderose**, « Cadre de référence pour l'amélioration de la qualité d'un logiciel à base de connaissances (application à COSMICXpert et SMXpert.) » - *Namur : Facultés universitaires Notre-Dame de la Paix, 2005.*
[14] **P Heymans**, « Analyse et modélisation de système d'information, INFO02107 » - *Namur : Facultés universitaires Notre-Dame de la Paix, Année académique 2004-2005*.

[*Wiki01*] http://fr.wikipedia.org/wiki/Raisonnement
[*Wiki02*] http://fr.wikipedia.org/wiki/Syst%C3%A8me_expert
[*Wiki03*] http://fr.wikipedia.org/wiki/R%C3%A9tro-ing%C3%A9nierie
[*Wiki04*] http://fr.wikipedia.org/wiki/MySQL

[*Web01*] http://tecfa.unige.ch/tecfa/publicat/schneider/these-daniel/wmwork/www/phd_45.html
[*Web02*] http://www.tripalium.com/fiches/auto-evaluation/sysexpert.html
[*Web03*] http://tecfa.unige.ch/tecfa/publicat/schneider/these-daniel/wmwork/www/phd_48.html
[*Web04*] http://villemin.gerard.free.fr/LogForm/Raisonne.htm
[*Web05*] http://www.strassmann.com/pubs/reengineering.html
[*Web06*] http://www.change-management.com/change-management-overview.htm
[*Web07*] http://www.digifactory.fr/prod/technologies.php?docid=310
[*Web08*] http://wiki.metacites.net/mysql